


# NFS Client How-To

## NFS Client How-To

Last Updated: June 18, 2012

### Table of Contents

-  [Adding NFS to the NuttX Configuration](#)
-  [Mount Interface](#)
-  [NFS Mount Command](#)
-  [Configuring the NFS server \(Ubuntu\)](#)

## Adding NFS to the NuttX Configuration

The NFS client is easily added to your configuration: You simply need to add `CONFIG_NFS` to your `nuttX/.config` file. There are, however, a few dependencies on other system settings:

1. First, there are things that you must configure in order to be able to use any file system:
  - `CONFIG_DISABLE_MOUNTPOINT=n`. You must include support for mount points in the pseudo-file system.
2. And there are several dependencies on the networking configuration. At a minimum, you need to have the following selections:
  - `CONFIG_NET=y`. General networking support.
  - `CONFIG_NET_UDP=y`. Support for UDP.

## Mount Interface

A low-level, C-callable interface is provided to mount a file system. That interface is called `mount()` and is mentioned in the [porting guide](#) and is prototyped in the header file `include/sys/mount.h`:

```
int mount(const char *source, const char *target, const char *filesystemtype, unsigned long mountflags,
const void *data);
```

**Synopsis:** `mount()` attaches the filesystem specified by the `source` block device name into the root file system at the path specified by `target`.

#### Input Parameters:

- `source`. A null-terminated string providing the full path to a block driver in the NuttX pseudo-file system.
- `target`. The location in the NuttX pseudo-file system where the volume will be mounted.
- `filesystemtype`. A string identifying the type of file system to use.
- `mountflags`. Various flags that can be used to qualify how the file system is mounted.
- `data`. Opaque data that is passed to the file system with the mount occurs.

**Returned Values** Zero is returned on success; -1 is returned on an error and `errno` is set appropriately:

- `EACCES`. A component of a path was not searchable or mounting a read-only filesystem was attempted without giving the `MS_RDONLY` flag.
- `EBUSY`. `source` is already mounted.
- `EFAULT`. One of the pointer arguments points outside the user address space.
- `EINVAL`. `source` had an invalid superblock.
- `ENODEV`. `filesystemtype` not configured
- `ENOENT`. A pathname was empty or had a nonexistent component.
- `ENOMEM`. Could not allocate a memory to copy filenames or data into.
- `ENOTBLK`. `source` is not a block device

This same interface can be used to mount a remote, NFS file system using some special parameters. The NFS mount differs from the *normal* file system mount in that: (1) there is no block driver for the NFS file system, and (2) special parameters must be passed as `data` to describe the remote NFS server. Thus the following code snippet might represent how an NFS file system is mounted:

```
#include <sys/mount.h>
#include <nuttX/fs/nfs.h>

struct nfs_args data;
char *mountpoint;

ret = mount(NULL, mountpoint, string "nfs", 0, (FAR void *)&data);
```

NOTE that: (1) the block driver parameter is `NULL`. The `mount()` is smart enough to know that no block driver is needed with the NFS file system. (2) The NFS file system is identified with the simple string "nfs" (3) A reference to `struct nfs_args` is passed as an NFS-specific argument.

The NFS-specific interface is described in the file `include/nuttx/fs/nfs.h`. There you can see that `struct nfs_args` is defined as:

```
struct nfs_args
{
    uint8_t  addrlen;           /* Length of address */
    uint8_t  sotype;           /* Socket type */
    uint8_t  flags;            /* Flags, determines if following are valid: */
    uint8_t  timeo;            /* Time value in deciseconds (with NFSMNT_TIMEO) */
    uint8_t  retrans;          /* Times to retry send (with NFSMNT_RETRANS) */
    uint16_t wsize;            /* Write size in bytes (with NFSMNT_WSIZE) */
    uint16_t rsize;            /* Read size in bytes (with NFSMNT_RSIZE) */
    uint16_t readdirsize;      /* readdir size in bytes (with NFSMNT_READDIRSIZE) */
    char     *path;            /* Server's path of the directory being mount */
    struct   sockaddr_storage addr; /* File server address (requires 32-bit alignment) */
};
```

## NFS Mount Command

The [NuttShell \(NSH\)](#) also supports a command called `nfsmount` that can be used to mount a remote file system via the NSH command line.

### Command Syntax:

```
nfsmount <server-address> <mount-point> <remote-path>
```

**Synopsis.** The `nfsmount` command mounts a network file system in the NuttX pseudo filesystem. The `nfsmount` will use NFSv3 UDP protocol to mount the remote file system.

**Command Line Arguments.** The `nfsmount` takes three arguments:

1. The `<server-address>` is the IP address of the server exporting the file system you wish to mount. This implementation of NFS for the NuttX RTOS is only for a local area network, so the server and client must be in the same network.
2. The `<mount-point>` is the location in the NuttX pseudo filesystem where the mounted volume will appear. This mount point can only reside in the NuttX pseudo filesystem. By convention, this mount point is a subdirectory under `/mnt`. The mount command will create whatever pseudo directories that may be needed to complete the full path (but the full path must not already exist).
3. The `<remote-path>` is the file system / directory being exported from server. This / directory must have been configured for exportation on the server before when the NFS server was set up.

After the volume has been mounted in the NuttX pseudo filesystem, it may be access in the same way as other objects in the file system.

**Example.** Suppose that the NFS server has been configured to export the directory `/export/shared`. The the following command would mount that file system (assuming that the target also has privileges to mount the file system).

```
NuttShell (NSH)
nsh> ls /mnt
/mnt:
nsh: ls: no such directory: /mnt
nsh> nfsmount 10.0.0.1 /mnt/nfs /export/shared
nsh> ls -l /mnt/nfs
/mnt/nfs:
drwxrwxrwx  4096 ..
drwxrwxrwx  4096 testdir/
-rw-rw-rw-    6 ctest.txt
-rw-r--r--   15 btest.txt
drwxrwxrwx  4096 .
nsh> echo "This is a test" >/mnt/nfs/testdir/testfile.txt
nsh> ls -l /mnt/nfs/testdir
/mnt/nfs/testdir:
-rw-rw-rw-    21 another.txt
drwxrwxrwx  4096 ..
drwxrwxrwx  4096 .
-rw-rw-rw-   16 testfile.txt
nsh> cat /mnt/nfs/testdir/testfile.txt
This is a test
```

## Configuring the NFS server (Ubuntu)

Setting up the server will be done in two steps: First, setting up the configuration file for NFS, and then starting the NFS services. But first, you need to install the `nfs` server on Ubuntu with these two commands:

```
# sudo apt-get install nfs-common
# sudo apt-get install nfs-kernel-server
```

After that, we need to make or choose the directory we want to export from the NFS server. In our case, we are going to make a new directory called `/export`.

```
# sudo mkdir /export
```

It is important that `/export` directory allow access to everyone (777 permissions) as we will be accessing the NFS share from the client with no authentication.

```
# sudo chmod 777 /export
```

When all this is done, we will need to edit the configuration file to set up an NFS server: `/etc/exports`. This file contains a list of entries; each entry indicates a volume that is shared and how it is shared. For more information for a complete description of all the setup options for this file you can check in the man pages (`man exports`).

An entry in `/etc/exports` will typically look like this:

```
directory machine1(option11,option12)
```

So for our example we export `/export` to the client 10.0.0.2 add the entry:

```
/export 10.0.0.2(rw)
```

In our case we are using all the default options except for the `ro` that we replaced with `rw` so that our client will have read and write access to the directory that we are exporting.

After we do all the require configurations, we are ready to start the server with the next command:

```
# sudo /etc/init.d/nfs-kernel-server start
```

Note: If you later decide to add more NFS exports to the `/etc/exports` file, you will need to either restart NFS daemon or run command `exportfs`.

```
# sudo /etc/init.d/nfs-kernel-server start
```

Or

```
# exportfs -ra
```

Now we can check if the export directory and our mount point is properly set up.

```
# sudo showmount -e
# sudo showmount -a
```

And also we can verify if NFS is running in the system with:

```
# rpcinfo -p
program vers proto  port
100000  2  tcp    111  portmapper
100000  2  udp    111  portmapper
100011  1  udp    749  rquotad
100011  2  udp    749  rquotad
100005  1  udp    759  mountd
100005  1  tcp    761  mountd
100005  2  udp    764  mountd
100005  2  tcp    766  mountd
100005  3  udp    769  mountd
100005  3  tcp    771  mountd
100003  2  udp    2049 nfs
100003  3  udp    2049 nfs
300019  1  tcp    830  amd
300019  1  udp    831  amd
100024  1  udp    944  status
100024  1  tcp    946  status
100021  1  udp    1042 nlockmgr
100021  3  udp    1042 nlockmgr
100021  4  udp    1042 nlockmgr
100021  1  tcp    1629 nlockmgr
100021  3  tcp    1629 nlockmgr
100021  4  tcp    1629 nlockmgr
```

Now your NFS sever is sharing `/export` directory to be accessed.