# NFSv4 Installation Guide

**Giorgos Kappes <geokapp@gmail.com>**

***Last update: 18 August 2011***

---

## 1. Introduction

The Network File System (NFS) was originally developed by SUN Microsystems to allow machines to mount a disk partition on a remote machine. The machines providing the service (in this case the file system) are the servers and the machines using the service are the clients. Files residing physically on the server appear as if they are local to the client. This allows for fast, seamless sharing of files across a network.

Currently, there are three versions of NFS. NFS version 2 (NFSv2) is the oldest and only allow clients to access data with a maximum size of 2GB. NFS version 3 (NFSv3) supports 64-bit file sizes and offsets, to handle files larger than 2 gigabytes (GB). It also supports asynchronous writes on the server, to improve write performance. These versions of NFS are stateless which means that the file server stores no per-client information, and there are no NFS "connections". The last version of NFS is NFS version 4 (NFSv4). Unlike the previous versions, the new version introduces a stateful and more secure protocol. Furthermore, the new version can work through the internet and provides many performance improvements.

NFS implementation on Linux consists of two parts:

- Kernel code under the directory 'fs/' in Linux kernel source tree:
    1. 'fs/nfs'
    2. 'fs/nfs-common'
    3. 'fs/nfsd'
- Server/client code and tools

In the remainder of this guide we will explain how to build and setup an NFSv4 server and how to configure it in order to export a file system for the clients.

## 2. Preparation

Our working environment is Debian Squeeze, so some of the following instructions may differ slightly if you are using a non-Debian based Linux distribution. In order to build

and use NFSv4 we will need a recent (2.6 or newer) Linux kernel source tree. For this guide we are going with the latest stable Linux kernel which is version 3.0. We can download and extract a Linux kernel source tree with the following commands:

```
root@jurassic# cd /usr/src
root@jurassic# wget http://www.kernel.org/pub/linux/kernel/v3.0/linux-3.0.tar.gz
root@jurassic# tar xvf linux-3.0.tar.gz
```

The server/client code and all the necessary utilities are included in the 'nfs-utils' package. You can obtain the source of this package through the CITI page: http://www.citi.umich.edu/projects/nfsv4/linux/ or through the Debian source repositories as shown below:

```
root@jurassic# apt-get source nfs-utils
```

In this guide we follow the second road and we download the version '1.2.2' of 'nfs-utils' from Debian Squeeze source repository. In order to successfully build this package we will need some additional packages and tools:

- GCC and make (in Debian you can just download: build-essential )
- libtirpc-dev
- libcap-dev
- libwrap0-dev
- libblkid-dev
- libnfsidmap-dev
- libevent-dev
- libkeyutils-dev
- libldap2-dev

If you want GSS/Kerberos support download and install the following packages too:

- libkrb5-dev
- librpcsecgss-dev
- libgssglue-dev

## 3. NFS Server

In this section we will explain how to build and configure an NFS server. We will start from building a new Linux kernel with NFS support and then we will show how to build

the required services and utilities contained in the 'nfs-utils' package. Finally, we will show how to configure our server to export a file system to some clients.

## 3.1. Kernel configuration

In order to use NFSv4 we'll need a recent Linux kernel (2.6 or newer) with the following options enabled as build-in:

```
Network File Systems →
        NFS client support – enabled
        NFS client support for NFS version 3 – enabled
        NFS client support for the NFSv3 ACL protocol expansion
        – enabed
        NFS client support for NFS version 4 – enabled
        Use the new idmapper upcall routine – enabled
        NFS server support – enabled
        NFS server support for NFS version 3 – enabled
        NFS server support for the NFSv3 ACL protocol expansion
        – enabed
        NFS server support for NFS version 4 – enabled
```

Assuming our server's file system is ext4, we should also select the following options if we wish to use NFSv4's extended attributes and ACL features:

```
File Systems →
        The extended 4 (ext4) filesystem
                Ext4 Extended attributes – enabled
                Ext4 POSIX Access Control Lists – enabled
```

If our current kernel does not meet the above requirements then we have to build and install a new kernel with the above options enabled. Then, we can boot to the new kernel.

## 3.2. NFS services and utilities

Having booted to an NFS enabled Linux kernel we can proceed on installing the necessary NFS services and utilities. As we said earlier, the package that contains these services and utilities is the 'nfs-utils'. If you don't wish to do any modifications to the source code of 'nfs-utils' you can simply install it using the following command as root:

```
root@jurassic# apt-get install nfs-utils rpcbind
```

This command will download and install 'nfs-utils' and all the required dependencies. Although the above command will do the trick, in this guide we focus on how to build 'nfs-utils' from source. Firstly, we must download all the required packages:

```
root@jurassic# apt-get install build-essential libtirpc-dev libcap-dev libwrap0-dev
libblkid-dev libnfsidmap-devlibevent-dev libkeyutils-devlibldap2-dev rpcbind
```

Next, we must download the 'nfs-utils' package. We can download it from the CITI page (see references) or from Debian sources with the following command:

```
root@jurassic# apt-get source nfs-utils
```

If we download 'nfs-utils' from CITI we get the 1.1.2 version. Instead, if we download it from Debian sources we get the 1.2.2 version. Now, we can proceed with the building:

```
root@jurassic# cd nfs-utils-1.x.x
root@jurassic# ./configure --disable-gss
root@jurassic# make
root@jurassic# make install
```

These commands will install mountd, mount.nfs, mount.nfs4, idmapd and exportfs. Note, that in the 'configure' script we have specified the '--disable-gss parameter'. This parameter disables the NFSv4 support for GSS/Kerberos. If you wish to use them install the required packages mentioned on section 2 and remove this parameter from the 'configure' script.

The next step is to create the '/var/lib/nfs/v4recovery' directory and the mount point for rpc_pipefs:

```
root@jurassic# mkdir /var/lib/nfs/v4recovery
root@jurassic# mkdir /var/lib/nfs/rpc_pipefs
```

After that, we must add the following entries to the '/etc/fstab' file:

```
rpc_pipefs  /var/lib/nfs/rpc_pipefs   rpc_pipefs   defaults   0 0
nfsd        /proc/fs/nfsd             nfsd         defaults   0 0
```

In order for the ID names to be automatically mapped, both the server and the clients require the '/etc/idmapd.conf' file to have the same contents with the correct domain names. However, the clients may have different requirements for the 'Nobody-User' and 'Nobody-Group'. This way, server and clients do not need the users to share the same UIDs/GIDs. We create the file '/etc/idmapd.conf' and write the following lines:

```
[General]

Verbosity = 0
Pipefs-Directory = /var/lib/nfs/rpc_pipefs
Domain = localdomain

[Mapping]

Nobody-User = nobody
Nobody-Group = nogroup
```

We will need to create four additional files. First, we create the file '/etc/default/nfs-common' and we add the following lines to it:

```
# If you do not set values for the NEED_ options, they will be
attempted
# autodetected; this should be sufficient for most people. Valid
alternatives
# for the NEED_ options are "yes" and "no".

# Do you want to start the statd daemon? It is not needed for NFSv4.
NEED_STATD=no

# Options for rpc.statd.
#   Should rpc.statd listen on a specific port? This is especially useful
#   when you have a port-based firewall. To use a fixed port, set this
#   this variable to a statd argument like: "--port 4000 --outgoing-port
4001".
# For more information, see rpc.statd(8) or
http://wiki.debian.org/?SecuringNFS
STATDOPTS=

# Do you want to start the idmapd daemon? It is only needed for NFSv4.
NEED_IDMAPD=yes

# Do you want to start the gssd daemon? It is required for Kerberos
mounts.
NEED_GSSD=no
```

Note, that we have disabled the 'NEED_STATD' since the NFSv4 protocol does not need the 'statd' daemon to work. Furthermore, we have enabled the 'NEED_IDMAPD' since NFSv4 needs the 'idmapd' daemon. Finally, we have disabled the option 'NEED_GSSD' because we don't wish to use GSS/Kerberos. If you would like to use them, change this option to 'yes'.

The next file we must create is '/etc/default/nfs-kernel-server'. After creating this file, we write the following lines to it:

```
# Number of servers to start up
RPCNFSDCOUNT=8
```

```
# Runtime priority of server (see nice(1))
RPCNFSDPRIORITY=0

# Options for rpc.mountd.
# If you have a port-based firewall, you might want to set up
# a fixed port here using the --port option. For more information,
# see rpc.mountd(8) or http://wiki.debian.org/?SecuringNFS
RPCMOUNTDOPTS=--manage-gids

# Do you want to start the svcgssd daemon? It is only required for
Kerberos
# exports. Valid alternatives are "yes" and "no"; the default is "no".
NEED_SVCGSSD=no

# Options for rpc.svcgssd.
RPCSVCGSSDOPTS=no
```

Again, we have disabled support for GSS/Kerberos. Now we have to create a script in order for our NFS service to start automatically when the system boots. We create the file '/etc/inin.t/nfs-kernel-server' and chmod it to '755'. Then, we add the following lines to it:

```
#!/bin/sh

### BEGIN INIT INFO
# Provides:          nfs-kernel-server
# Required-Start:    $remote_fs nfs-common $rpcbind $time
# Required-Stop:     $remote_fs nfs-common $rpcbind $time
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Kernel NFS server support
# Description:       NFS is a popular protocol for file sharing across
#                    TCP/IP networks. This service provides NFS server
#                    functionality, which is configured via the
#                    /etc/exports file.
### END INIT INFO

# What is this?
DESC="NFS kernel daemon"
PREFIX=/usr

# Exit if required binaries are missing.
[ -x $PREFIX/sbin/rpc.nfsd    ] || exit 0
[ -x $PREFIX/sbin/rpc.mountd  ] || exit 0
[ -x $PREFIX/sbin/exportfs    ] || exit 0

# Read config
DEFAULTFILE=/etc/default/nfs-kernel-server
RPCNFSDCOUNT=8
RPCNFSDPRIORITY=0
RPCMOUNTDOPTS=
NEED_SVCGSSD=no
RPCSVCGSSDOPTS=
```

```
PROCNFSD_MOUNTPOINT=/proc/fs/nfsd
if [ -f $DEFAULTFILE ]; then
    . $DEFAULTFILE
fi

. /lib/lsb/init-functions



do_mount() {
    if ! grep -E -qs "$1\$" /proc/filesystems
    then
    return 1
    fi
    if ! mountpoint -q "$2"
    then
    mount -t "$1" "$1" "$2"
    return
    fi
    return 0
}

# See how we were called.
case "$1" in
  start)
    if [ -f /etc/exports ]
    then

            # See if our running kernel supports the NFS kernel server
            if ! grep -E -qs "[[:space:]]nfsd\$" /proc/filesystems; then
                    log_warning_msg "Not starting $DESC: no support in
current kernel."
                    exit 0
            fi

            do_mount nfsd $PROCNFSD_MOUNTPOINT ||
NEED_SVCGSSD=no
            log_begin_msg "Exporting directories for $DESC..."
            $PREFIX/sbin/exportfs -r
            RET=$?
            if [ $RET != 0 ]; then
                    log_end_msg $RET
                    exit $RET
            fi
            log_end_msg 0

            log_daemon_msg "Starting $DESC"
            log_progress_msg "nfsd"
            start-stop-daemon --start --oknodo --quiet \
               --nicelevel $RPCNFSDPRIORITY \
               --exec $PREFIX/sbin/rpc.nfsd -- $RPCNFSDCOUNT
            RET=$?
            if [ $RET != 0 ]; then
                    log_end_msg $RET
                    exit $RET
```

```
        fi

        # make sure 127.0.0.1 is a valid source for requests
        ClearAddr=
        if [ -f /proc/net/rpc/auth.unix.ip/channel ]
        then
           fgrep -qs 127.0.0.1 /proc/net/rpc/auth.unix.ip/content || {
                echo "nfsd 127.0.0.1 2147483647 localhost"
>/proc/net/rpc/auth.unix.ip/channel
                ClearAddr=yes
           }
        fi

        $PREFIX/bin/rpcinfo -u localhost nfs 3 >/dev/null 2>&1 ||
           RPCMOUNTDOPTS="$RPCMOUNTDOPTS --no-nfs-
version 3"

        [ -z "$ClearAddr" ] || echo "nfsd 127.0.0.1 1"
>/proc/net/rpc/auth.unix.ip/channel

       if [ "$NEED_SVCGSSD" = "yes" ]; then
                do_modprobe rpcsec_gss_krb5
                log_progress_msg "svcgssd"
                start-stop-daemon --start --oknodo --quiet \
                   --exec $PREFIX/sbin/rpc.svcgssd --
$RPCSVCGSSDOPTS
                RET=$?
                if [ $RET != 0 ]; then
                        log_end_msg $RET
                        exit $RET
                fi
        fi

        log_progress_msg "mountd"
        start-stop-daemon --start --oknodo --quiet \
           --exec $PREFIX/sbin/rpc.mountd -- $RPCMOUNTDOPTS
        RET=$?
        if [ $RET != 0 ]; then
                log_end_msg $RET
                exit $RET
        fi

        log_end_msg 0
  else
        log_warning_msg "Not starting $DESC: no exports."
  fi
  ;;

 stop)
  log_daemon_msg "Stopping $DESC"

  log_progress_msg "mountd"
  start-stop-daemon --stop --oknodo --quiet \
     --name rpc.mountd --user 0
  RET=$?
```

```
  if [ $RET != 0 ]; then
          log_end_msg $RET
          exit $RET
  fi

  if [ "$NEED_SVCGSSD" = "yes" ]; then
          log_progress_msg "svcgssd"
          start-stop-daemon --stop --oknodo --quiet \
             --name rpc.svcgssd --user 0
          RET=$?
          if [ $RET != 0 ]; then
                  log_end_msg $RET
                  exit $RET
          fi
  fi

  log_progress_msg "nfsd"
  start-stop-daemon --stop --oknodo --quiet \
     --name nfsd --user 0 --signal 2
  RET=$?
  if [ $RET != 0 ]; then
          log_end_msg $RET
          exit $RET
  fi

  log_end_msg 0

  log_begin_msg "Unexporting directories for $DESC..."
  $PREFIX/sbin/exportfs -au
  RET=$?
  if [ $RET != 0 ]; then
          log_end_msg $RET
          exit $RET
  fi
  log_end_msg 0

  if mountpoint -q $PROCNFSD_MOUNTPOINT
  then
     $PREFIX/sbin/exportfs -f
  fi
  ;;

 status)
  if pidof nfsd >/dev/null
  then
     echo "nfsd running"
     exit 0
  else
     echo "nfsd not running"
     exit 3
  fi
  ;;

 reload | force-reload)
  log_begin_msg "Re-exporting directories for $DESC..."
```

```
    $PREFIX/sbin/exportfs -r
    RET=$?
    log_end_msg $RET
    exit $RET
    ;;

  restart)
    $0 stop
    sleep 1
    $0 start
    ;;

  *)
    log_success_msg "Usage: nfs-kernel-server
{start|stop|status|reload|force-reload|restart}"
    exit 1
    ;;
esac

exit 0
```

After that, we have to create a script for nfs-common which contains the required NFS utilities. We create the file '/etc/init.d/nfs-common' and we chmod it to '755'. Then, we write the following lines to it:

```
 #!/bin/sh

### BEGIN INIT INFO
# Provides:          nfs-common
# Required-Start:    $rpcbind $time
# Required-Stop:     $time
# Default-Start:     2 3 4 5 S
# Default-Stop:      0 1 6
# Short-Description: NFS support files common to client and server
# Description:       NFS is a popular protocol for file sharing across
#                    TCP/IP networks. This service provides various
#                    support functions for NFS mounts.
### END INIT INFO

# What is this?
DESC="NFS common utilities"

# Read config
DEFAULTFILE=/etc/default/nfs-common
PREFIX=
NEED_STATD=
NEED_IDMAPD=
NEED_GSSD=
PIPEFS_MOUNTPOINT=/var/lib/nfs/rpc_pipefs
RPCGSSDOPTS=
if [ -f $DEFAULTFILE ]; then
    . $DEFAULTFILE
fi
```

```
. /lib/lsb/init-functions


#
# Parse the fstab file, and determine whether we need idmapd and gssd.
(The
# /etc/defaults settings, if any, will override our autodetection.) This
code
# is partially adapted from the mountnfs.sh script in the sysvinit package.
#
AUTO_NEED_IDMAPD=no
AUTO_NEED_GSSD=no

if [ -f /etc/fstab ]; then
    exec 9<&0 </etc/fstab

    while read DEV MTPT FSTYPE OPTS REST
    do
    case $DEV in
        "|\#*)
                continue
                ;;
    esac
        if [ "$FSTYPE" = "nfs4" ]; then
            AUTO_NEED_IDMAPD=yes
        fi
        case "$OPTS" in




sec=krb5|*,sec=krb5|sec=krb5,*|*,sec=krb5i,*|sec=krb5i|*,sec=krb5i|
sec=krb5i,*|*,sec=krb5i,*|sec=krb5p|*,sec=krb5p|sec=krb5p,*|*,sec=k
rb5p,*)
                AUTO_NEED_GSSD=yes
            ;;
        esac
    done

    exec 0<&9 9<&-
fi

#
# We also need idmapd if we run an NFSv4 server. It's fairly difficult
# to autodetect whether there are NFSv4 exports or not, and idmapd is
not a
# particularily heavy daemon, so we auto-enable it if we find an
/etc/exports
# file. This does not mean that there are NFSv4 or other mounts active
(or
# even that nfs-kernel-server is installed), but it matches what the "start"
# condition in nfs-kernel-server's init script does, which has a value in
# itself.
#
if [ -f /etc/exports ] && grep -q '^[[:space:]]*[^#]*/' /etc/exports; then
```

```
    AUTO_NEED_IDMAPD=yes
fi

case "$NEED_STATD" in
    yes|no)
        ;;
    *)
        NEED_STATD=yes
        ;;
esac

case "$NEED_IDMAPD" in
    yes|no)
        ;;
    *)
        NEED_IDMAPD=$AUTO_NEED_IDMAPD
    ;;
esac

case "$NEED_GSSD" in
    yes|no)
        ;;
    *)
        NEED_GSSD=$AUTO_NEED_GSSD
    ;;
esac

do_modprobe() {
    if [ -x /sbin/modprobe -a -f /proc/modules ]
    then
        modprobe -q "$1" || true
    fi
}

do_mount() {
    if ! grep -E -qs "$1\$" /proc/filesystems
    then
    return 1
    fi
    if ! mountpoint -q "$2"
    then
    mount -t "$1" "$1" "$2"
    return
    fi
    return 0
}

do_umount() {
    if mountpoint -q "$1"
    then
    umount "$1"
    fi
    return 0
}
```

```
# See how we were called.
case "$1" in
  start)
    log_daemon_msg "Starting $DESC"

    if [ "$NEED_STATD" = yes ]; then
       log_progress_msg "statd"
       start-stop-daemon --start --oknodo --quiet \
          --exec $PREFIX/sbin/rpc.statd -- $STATDOPTS
       RET=$?
       if [ $RET != 0 ]; then
          log_end_msg $RET
          exit $RET
       fi
    fi

    # Don't start idmapd and gssd if we don't have them (say, if /usr is
not
    # up yet).
    [ -x /usr/sbin/rpc.idmapd ] || NEED_IDMAPD=no
    [ -x /usr/sbin/rpc.gssd   ] || NEED_GSSD=no

    if [ "$NEED_IDMAPD" = yes ] || [ "$NEED_GSSD" = yes ]
    then
       do_modprobe sunrpc
       if do_mount rpc_pipefs $PIPEFS_MOUNTPOINT
       then
            if [ "$NEED_IDMAPD" = yes ]
            then
             log_progress_msg "idmapd"
               start-stop-daemon --start --oknodo --quiet \
                       --exec /usr/sbin/rpc.idmapd
               RET=$?
             if [ $RET != 0 ]; then
                  log_end_msg $RET
                  exit $RET
                fi
            fi
            if [ "$NEED_GSSD" = yes ]
            then
               do_modprobe rpcsec_gss_krb5
             log_progress_msg "gssd"

             # we need this available; better to fail now than
               # mysteriously on the first mount
               if ! grep -q -E '^nfs[         ]' /etc/services; then
                    log_action_end_msg 1 "broken /etc/services, please see
/usr/share/doc/nfs-common/README.Debian.nfsv4"
                    exit 1
               fi

               start-stop-daemon --start --oknodo --quiet \
                       --exec /usr/sbin/rpc.gssd -- $RPCGSSDOPTS
               RET=$?
             if [ $RET != 0 ]; then
```

```
                log_end_msg $RET
                exit $RET
              fi
            fi
        fi
    fi
    log_end_msg 0
    ;;

stop)
    log_daemon_msg "Stopping $DESC"

    if [ "$NEED_GSSD" = yes ]
    then
        log_progress_msg "gssd"
        start-stop-daemon --stop --oknodo --quiet \
                --name rpc.gssd
        RET=$?
        if [ $RET != 0 ]; then
            log_end_msg $RET
            exit $RET
        fi
    fi
    if [ "$NEED_IDMAPD" = yes ]
    then
        log_progress_msg "idmapd"
        start-stop-daemon --stop --oknodo --quiet \
            --name rpc.idmapd
        RET=$?
        if [ $RET != 0 ]; then
            log_end_msg $RET
            exit $RET
        fi
    fi
    if [ "$NEED_STATD" = yes ]
    then
        log_progress_msg "statd"
        start-stop-daemon --stop --oknodo --quiet \
            --name rpc.statd
        RET=$?
        if [ $RET != 0 ]; then
            log_end_msg $RET
            exit $RET
        fi
    fi
    do_umount $PIPEFS_MOUNTPOINT 2>/dev/null || true
    log_end_msg 0
    ;;

status)
    if [ "$NEED_STATD" = yes ]
    then
        if ! pidof rpc.statd >/dev/null
        then
            echo "rpc.statd not running"
```

```
            exit 3
        fi
    fi

    if [ "$NEED_GSSD" = yes ]
    then
            if ! pidof rpc.gssd >/dev/null
        then
                echo "rpc.gssd not running"
                exit 3
        fi
    fi

    if [ "$NEED_IDMAPD" = yes ]
    then
            if ! pidof rpc.idmapd >/dev/null
        then
                echo "rpc.idmapd not running"
                exit 3
        fi
    fi

    echo "all daemons running"
    exit 0
    ;;

  restart | force-reload)
    $0 stop
    sleep 1
    $0 start
    ;;

  *)
    log_success_msg "Usage: nfs-common {start|stop|status|restart}"
    exit 1
    ;;
esac

exit 0
```

After creating these init scripts we have to register them to the system. We run the following commands:

```
root@jurassic# cd /etc/init.d
root@jurassic# update-rc.d nfs-common defaults
root@jurassic# update-rc.d nfs-kernel-server defaults
```

## 3.3. Exporting directories

After completing the above steps our NFS server is almost ready to be used. A final configuration that we must do is to specify a file system to export to clients. Firtsly, we create the file '/etc/exports' with the following content:

```
# /etc/exports: the access control list for filesystems which may be
exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check)
hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#
```

Note that the above lines are just comments. In this file we specify which directories we would like to export to which clients. Also, we can specify some export options. Each entry for an exported file system has the following structure (if we don't use GSS/Kerberos):

```
export host/ip/network(options)
```

Where:

- export: is the directory being exported. (/exports)
- host/ip/network: is the host/ ip address or network where we would like to export that directory. Some examples are: zeus, 192.168.1.20, 192.168.1.0/24, 192.168.1.0/255.255.255.0.
- options: The options to be used for the exported file system. The default options while exporting a directory are:
    - **'ro'**: The exported file system is read-only. Remote hosts cannot change its data. To allow remote hosts to make changes to the file system, specify the **'rw'** option.
    - **'sync'**: The NFS server will not reply to requests before changes made by previous requests are written to disk. To enable asynchronous writes specify the option: **'async'**.
    - **'wdelay'**: The NFS server will delay writing to the disk if it suspects another write request is imminent. To disable this, specify the **'no_wdelay'** option. Note that 'no_wdelay' is only available if the the 'sync' option is also specified.

- **'root_squash'**: This option prevents root users connected remotely from having root privileges in the exported file system. Instead, the NFS server will assign them the 'nfsnobody' user id. To disable root squashing, specify **'no_root_squash'**. To squash every remote user use the option **'all_squash'**.
- **'secure'**: This option requires that requests originate on an internet port less than 'IPPORT_RESERVED' (1024). This option is on by default. To turn it off, specify **'insecure'**.

There are some additional interest options except from the above:

- **'fsid=0'**: An NFSv4 client now has the ability to see all of the exports served by the NFSv4 server as a single file system, called the NFSv4 pseudo-file system. The pseudo-file system is identified as a single, real file system, identified at export with the 'fsid=0' option. The 'fsid=0' signals that the current export is the root.
- **'no_subtree_check'**: When an NFS server exports a subdirectory of a local file system, but leaves the rest unexported, the NFS server must check whether each NFS request is against a file residing in the area that is exported. This check is called the subtree check. To perform this check, the server includes information about the parent directory of each file in NFS file handles that are handed out to NFS clients. If the file is renamed to a different directory, for example, this changes the file handle, even though the file itself is still the same file. This breaks NFS protocol-compliance, often causing misbehavior on clients such as 'ESTALE' errors, inappropriate access to renamed or deleted files, broken hard links, and so on. The 'no_subtree_check' option disables this check.

Now, let's assume that we want to export our '/home' directory. First, we have to create the export directory:

```
root@jurassic# mkdir /export
root@jurassic# chmod a+rwxt /export
```

And mount the real '/home' directory with:

```
root@jurassic# mkdir –p /export/home
root@jurassic# mount –bind /home /export/home
```

If we would like the '/home' directory to be mounted automatically on the export directory upon boot, we can add the following line to the '/etc/fstab':

```
/home           /export/home        none        bind    0       0
```

Let's assume that we would like to export the '/export/home' as the root directory to the network 192.168.2.0/24. We can add the following line to the '/etc/exports':

```
/export
192.168.2.0/24(rw,fsid=0,wdelay,insecure,no_subtree_check,async)
```

In order for this to take effect we must reboot our system. Although, if the NFS server was already running we can just run the following command to export the new directories which we added in the '/etc/exports' file:

```
root@jurassic# exportfs –a
```

# 4. NFS Client

In this section we will explain how to configure an NFS client. We will start from building an NFS-enabled kernel and then we will see how to build and install the required utilities. Finally we will try to mount the exported file system from the server.

## 4.1. Kernel configuration

In order for a client to be able to mount a file system exported from an NFSv4 server it requires a 2.6 or more recent Linux kernel with the following options enabled:

```
Network File Systems →
        NFS client support – enabled
        NFS client support for NFS version 3 – enabled
        NFS client support for the NFSv3 ACL protocol expansion
        – enabed
        NFS client support for NFS version 4 – enabled
        Use the new idmapper upcall routine – enabled
```

Furthermore, if we wish the client to be able to mount its root over NFS we must enable some additionally options in the kernel:

```
Networking support →
        Networking options →
                IP: kernel level autoconfiguration – enabled
                IP: DHCP support – enabled
                IP: BOOTP support – enabled
                IP: RARP support – enabled
```

```
File Systems →
        Network File Systems →
                Root filesystem on NFS – enabled
```

## 4.2. NFS utilities

Like previously, we need the 'nfs-utils' package. If we don't wish to do any modifications to the source code of 'nfs-utils' we can simply install it using the following command as root:

```
root@helios# apt-get install nfs-common rpcbind
```

This command will download and install 'nfs-utils' and all the required dependencies. As we said in section 3.1 if we wish to build the NFS utilities from source we can download this package from CITI or from Debian source repositories with the following command:

```
root@helios# apt-get source nfs-utils
```

Before we start building it we must download and install its dependencies:

```
root@helios# apt-get install build-essential libtirpc-dev libcap-dev libwrap0-dev
libblkid-dev libnfsidmap-devlibevent-dev libkeyutils-devlibldap2-dev rpcbind
```

Now, we can proceed with the building:

```
root@helios# cd nfs-utils-1.x.x
root@helios# ./configure --disable-gss
root@helios# make
root@helios# make install
```

Note, that in the 'configure' script once more we have specified the '--disable-gss' parameter. If you wish to use GSS/Kerberos then install the required packages mentioned on section 2 and remove this parameter from the 'configure' script.

Now, we must create the '/var/lib/nfs/v4recovery' directory and the mount point for rpc_pipefs:

```
root@helios# mkdir /var/lib/nfs/v4recovery
root@helios# mkdir /var/lib/nfs/rpc_pipefs
```

After that, we add the following entries to the '/etc/fstab' file:

```
rpc_pipefs  /var/lib/nfs/rpc_pipefs  rpc_pipefs  defaults  0 0
nfsd        /proc/fs/nfsd            nfsd        defaults  0 0
```

It's time to create some additional files. The first one is '/etc/default/nfs-common'. The content of this file is the same as the corresponding file in section 3.2. The next file we must create is the '/etc/idmapd.conf'. Again, we add to this file the following content:

```
[General]

Verbosity = 0
Pipefs-Directory = /var/lib/nfs/rpc_pipefs
Domain = localdomain

[Mapping]

Nobody-User = nobody
Nobody-Group = nogroup
```

Note, that we must use the same domain with the server. Also, note that we must specify the correct nobody user/group which can be different from the server. We can look to '/etc/passwd' and '/etc/groups' files for them.

The final file we have to create is '/etc/init.d/nfs-common'. Again we add the same content with the corresponding file in section 3.2. Next we have to chmod it to '755' and run the following command in order to update the init scripts:

```
root@helios# update-rc.d nfs-common defaults
```

## 4.3. Mount server exports

We can mount the complete export tree with the following command:

```
root@helios# mount –t nfs4 –o proto=tcp,port=2049 nfs-server:/ /local/folder
```

Of course we can mount a specific exported subtree by specifying it in the mount command:

```
root@helios# mount –t nfs4 –o proto=tcp,port=2049 nfs-server:/some-directory
/local/folder
```

Also, we can add the following line to '/etc/fstab' in order for the remote file system to be mounted automatically on the boot. The 'auto' option mounts the exported file system on start up, while the '_netdev' option waits until the network devices are loaded:

```
nfs-server:/       /local/directory       nfs4    _netdev,auto  0       0
```

## 5. References


1. Center for Information Technology Integration [http://www.citi.umich.edu/projects/nfsv4/linux/]
2. NFSv4 Linux wiki [http://wiki.linux-nfs.org/wiki/index.php/Main_Page]
3. Securing NFS [http://wiki.debian.org/SecuringNFS]