NFS-Root mini-HOWTO

Table of Contents

NFS-Root mini-HOWTO

not maintained

V9, 20 September 2002

This mini-HOWTO tries explains how to set up a ``diskless'' Linux workstation, which mounts its root filesystems via NFS. The newest version of this mini-HOWTO can always be found at http://www.tldp.org/HOWTO/mini/NFS-Root.html *or a Linux Documentation Project mirror NEAR YOU.*

1. Copyright

If you have questions, please contact Andreas Kostyrka <`mailto:andreas@ag.or.at`>, the author of this mini-HOWTO, or the Linux HOWTO coordinator, at <`mailto:linux-howto@sunsite.unc.edu`> via email.

1.1 Contributors

- Avery Pennarun `<apenwarr @ foxnet.net>` (how to boot without **LILO**)
- Ofer Maor `<ofer @ hadar.co.il>` (a better mini-HOWTO about setting up diskless workstations)
- Christian Leutloff `<leutloff @ sundancer.tng.oche.de>` (info about netboot)
- Greg Roelofs `<newt @ pobox.com>` (2.2/2.4 updates, DHCP info, NFS-export info)

2. General Overview

An NFS-mounted root filesystem is typically most useful in two situations:

- A system administrator may wish to aggregate storage for multiple workstations in order to simplify maintenance, improve security and reliability, and/or make more economical use of limited storage capacity. In this scenario, a single, large server may host a dozen or more workstations; all of the

systems can be regularly backed up from a central location, and individual clients are less prone to damage by unsophisticated users or attack by malicious parties with physical access. (Of course, if the server itself is compromised, then so are all of the clients.)
- An embedded system may not have a disk, an IDE interface, or even a PCI bus. Even if it does, during development it may be too unstable to use the disk, and a ramdisk may be too small to include all of the necessary utilities or too large (as a part of the kernel image) to allow for rapid turnaround during testing and development. An NFS root allows quick kernel downloads, helps ensure filesystem integrity (since the server is basically impervious to crashes by the client), and provides virtually infinite storage.

(In this document we'll use the terms *client* and *workstation* interchangeably.)

However, there are two small problems from the client's perspective:

- It must find out its own IP address and possibly also the rest of the ethernet configuration (gateway, netmask, name servers, etc.).
- It must know or discover both the IP address of the NFS server and the mount path (on the server) to the exported root filesystem.

The current implementation of *NFSROOT* in the Linux kernel (as of 2.4.x) allows for several approaches, including:

- The complete ethernet configuration, including the *NFS*-path to be mounted, may be passed as parameters to the kernel via **LILO**, **LOADLIN**, or a hard-coded string within `linux/arch/i386/kernel/setup.c` (or its equivalent for other architectures).
- The IP address may be discovered by *RARP* and the *NFS*-path passed via kernel parameters.
- The IP address may be discovered by *RARP*, with the *NFS*-path derived from the *RARP* server and the just-granted IP address (loosely speaking, ``mount -t nfs <*RARP-server*>:/tftpboot/<*IP-address-of-client*>/dev/nfs'').
- The client configuration may be discovered by *BOOTP*.
- The client configuration may be discovered by *DHCP*.

Since the most common dynamic-address protocol these days is DHCP, its addition as an option in kernels 2.2.19 and 2.4.x (3 < x <= 14) is particularly welcome.

Before starting to set up a diskless environment, you should decide if you will be booting via **LILO**, **LOADLIN**, or a custom, embedded bootloader. The advantage of using something like **LILO** is flexibility; the disadvantage is speed--booting a Linux kernel without **LILO** is faster. This may or may not be a consideration.

3. Setup on the server

3.1 Compiling the kernels

On the server side, if you don't plan to use the old, user-mode NFS daemon, you'll need to compile NFS server support into the kernel (``NFS server support,'' a.k.a. *knfsd* or `CONFIG_NFSD`). If you plan to use the older *RARP* protocol to assign the client an IP address, *RARP* support in the kernel of the server is probably a good idea. (You must have it if you will boot via RARP without kernel parameters.) On the other hand, it doesn't help you if the client isn't on the same subnet as the server.

The kernel for the workstation needs the following settings, as a minimum:

- *NFS filesystem support* (`CONFIG_NFS_FS`). Note that there is no need for *ext2* support.
- *Root file system on NFS* (`CONFIG_ROOT_NFS`).
- *Ethernet (10 or 100Mbit)* (`CONFIG_NET_ETHERNET`).
- The ethernet driver for the workstation's network card (or onboard ethernet chip, if it's built into the motherboard or chipset).

Where there is an option to compile something in as a module, do *not* do so; modules only work *after* the kernel is booted, and these things are needed *during* boot.

For dynamically assigned IP numbers, you'll also need to select one or more of these kernel options:

- *IP: kernel level autoconfiguration* (`CONFIG_IP_PNP`)
- *RARP support* (`CONFIG_IP_PNP_RARP`)
- *BOOTP support* (`CONFIG_IP_PNP_BOOTP`)
- *DHCP support* (`CONFIG_IP_PNP_DHCP`)

If the workstation will be booted without kernel parameters, you need also to set the root device to 0:255. Do this by creating a dummy device file with `mknod /dev/nfsroot b 0 255`. After having created such a device file, you can set root device of the kernel image with `rdev <kernel-image> /dev/nfsroot`. [*NOTE: Modern kernels recognize `root=/dev/nfs` as a command-line argument; for consistency and/or compatibility, it may be better to use `/dev/nfs` as the device name instead of `/dev/nfsroot`.*]

3.2 Creation of the root filesystem

Copying the filesystem

Warning: while these instruction might work for you, they are by no means sensefull in a production environment. For a better way to set up a root filesystem for the clients, see the NFS-Root-Client mini-HOWTO by Ofer Maor <`ofer@hadar.co.il`>.

After having decided where to place the root tree, create it with (e.g.) `mkdir -p <directory>` and `tar cClf / - | tar xpCf <directory> -`.

If you boot your kernel without LILO, then the rootdir has to be `/tftpboot/<IP-address>`. If you don't like it, you can change it in the top Makefile in the kernel sources, look for a line like: `NFS_ROOT = -DNFS_ROOT="\"/tftpboot/%s\""` If you change this, you have to recompile the kernel.

Changes to the root filesystem

Now trim the unneeded files, and check the /etc/rc.d scripts. Some important points:

- One important thing is eth0 setup. The workstation comes up with eth0 set up, at least partially. Setting the IP address of the workstation to the IP address of the server is not considered a clever thing to do. (As it happened to the original author on one of his early attempts.)
- Another point is the /etc/fstab of the workstation. It should be set up for NFS filesystems. *<NOTE: this is not true in 2.4 kernels; the NFS mount is implicit and may actually cause mount(1) error messages if it's explicitly listed in /etc/fstab. It is not clear when this changed.>*

- **WARNING**: Don't confuse the server root filesystem and the workstation root filesystem. (I've already patched up a rc.inet1 on the server, and wondered why the workstation still didn't work.)

Exporting the filesystem

Export the root dir to the workstation. The basic idea is to edit `/etc/exports` to include a line similar to one of the following:

- `/path/on/server/to/nfs_root <client-IP-number>(rw,no_root_squash,no_all_squash) <2nd-client-IP-number>(rw,no_root_squash,no_all_squash)`
- `/path/on/server/to/nfs_root <client-IP-network>/<client-IP-netmask>(rw,no_root_squash,no_all_squash)`

For example, a DHCP client receiving an IP address on a class C subnet would need an exports entry similar to this:

- `/path/on/server/to/nfs_root 192.168.263.0/255.255.255.0(rw,no_root_squash,no_all_squash)`

The `no_root_squash` parameter allows the superuser (root) to be treated as such by the NFS server; otherwise *root* will be remapped to *nobody* and will generally be unable to do anything useful with the filesystem. The `no_all_squash` parameter is similar but applies to non-root users. See the `exports(5)` man page for details.

You will have to notify the NFS server after making any changes to the exports file. Under Red Hat this can easily be done by typing `/etc/rc.d/init.d/nfs stop; /etc/rc.d/init.d/nfs start`. On other systems, a simple `/etc/rc.d/init.d/nfs restart` or even `exportfs -a` may suffice, while on older machines running the user-mode NFS daemon you may actually need to `killall -HUP rpc.mountd; killall -HUP rpc.nfsd`. (Do *not* `killall -HUP rpc.portmap`, however!)

You may also need to edit `/etc/hosts.allow` and/or `/etc/hosts.deny` if tcp_wrappers are installed. In particular, if the remote system (client) gets *RPC: connection refused* errors, `/etc/hosts.deny` probably contains `portmap: ALL` or `ALL: ALL`. To enable the client to use the server's portmapper, add a corresponding line to `/etc/hosts.allow`:

```
portmap: <client-IP-number>
portmap: <2nd-client-IP-number>
portmap: <client-IP-network>/<client-IP-netmask>
```

There is no need to restart anything in this case. You can check by running `rpcinfo -p` on the NFS server and `rpcinfo -p` *`NFS-server`* on a Linux client within the allowed range; the RPC services listed by both should match.

In case of problems, check `/var/log/messages` and `/var/log/syslog` for errors (for example, run `tail -f /var/log/messages /var/log/syslog` and then try booting the client), and check your man pages (exports, exportfs, portmap, etc.). As a last resort, a reboot of the NFS server may help, but that's a borderline Microsoftism...

RARP setup

Set up the *RARP* somewhere on the net. If you boot without a nfsroot parameter, the *RARP* server has to be the *NFS* server. Usually this will be the *NFS* server. To do this, you will need to run a kernel with *RARP* support.

To do this, execute (and install it somewhere in `/etc/rc.d` of the server!):

```
/sbin/rarp -s <ip-addr> <hardware-addr>
```

where

ip-addr
: is the IP address of the workstation, and

hardware-addr
: is the ethernet address of the network card of the workstation.

example: `/sbin/rarp -s 131.131.90.200 00:00:c0:47:10:12`

You can also use a symbolic name instead of the IP address, as long the server is able to find out the IP address. (/etc/hosts or *DNS* lookups)

BOOTP setup

For *BOOTP* setup you need to edit `/etc/bootptab`. Please consult the *bootpd(8)* and *bootptab(5)* man pages.

DHCP setup

There is no need for the DHCP server to be the same as the NFS server, and in most cases, a DHCP server will already be set up. If one is not, however, consult the DHCP mini-HOWTO for further help.

Finding out hardware addresses

I don't know the hardware address! How can I find it out?

- Boot the kernel disk you made, and watch for the line where the network card is recognized. It usually contains 6 hex bytes, that should be the address of the card.
- Boot the workstation with some OS with TCP/IP networking enabled. Then ping the workstation from the server. Look in the ARP-cache by executing: `/sbin/arp -a`

4. Booting the workstation

4.1 Using a boot ROM

As I have not used such a beast myself yet, I can give you only the following tips (courtesy of Christian Leutloff `<leutloff@sundancer.tng.oche.de>`):

- You can't use ``normal'' boot ROMs.

- There is a `netboot` packet by Gero Kuhlmann, that provides for boot ROMs for Linux, and further information. `netboot` is available from the local Linux mirror, or as a Debian package (`netboot-0.4`).
- Read the documentation coming with your boot ROM carefully.
- You probably will have to enable the tftpd on the server, but this depends upon your boot ROM's way of loading the kernel.
- *Any information on boot-ROM vendors of these Linux variety, mentioned above, as not everybody has access to PROM burner :((especially in Europe, as I'm located there.) welcome, I'll include them then here.*

4.2 Using a raw kernel disk

If you have exported the root filesystem with the correct name for the default naming and your *NFS* server is also the *RARP* server (which implies that the boxes are on the same subnet.), than you can just boot the kernel by `cat`ing it to a disk. (You have to set the root device in the kernel to 0:255.) This assumes, that the root directory on the server is `/tftpboot/`*IP Address* (this value can be changed when compiling the kernel.)

4.3 Using a bootloader & *RARP*

Give the kernel all needed parameters when booting, and add `nfsroot=<`*`server-ip-addr`*`>:</`*`path/to/mount`*`>` where *server-ip-addr* is the IP address of your NFS-server, and */path/to/mount* is the path to the root directory.

Tips:

- When using **LILO** consider using the ``` ``lock" ``` feature: Simply type in once all the correct parameters and add ``` ``lock" ```. Next time when booting let LILO timeout.
- When generating a workstation specific boot disk, you can also use the `append=` feature in `lilo.conf`.

4.4 Using a bootloader without *RARP*

The `ip` and `nfsroot` kernel parameters (which can be hardcoded into the kernel, interactively entered at some bootloader prompts, or included in `lilo.conf` via the `append=` parameter; see the next subsection) provide all of the information necessary for the client to set up its ethernet interface and to contact the NFS server, respectively. The parameters are fully documented in `Documentation/nfsroot.txt`, which is included in the kernel sources (usually found under `/usr/src/linux`). Here's the format for a machine with a static (pre-assigned) IP address:

- `nfsroot=<NFS-server-IP-number>:/path/on/server/to/nfs_root ip=<client-IP-number>::<gateway-IP-number>:<netmask>:<client-hostname>:eth`

DHCP is much simpler:

- `nfsroot=<NFS-server-IP-number>:/path/on/server/to/nfs_root ip=dhcp`

4.5 Sample kernel command lines

Here's an example of a complete kernel command line such as you might include in `lilo.conf` or equivalent; only the IP numbers and NFS path are bogus:

- `root=/dev/nfs rw nfsroot=12.345.67.89:/path/on/server/to/nfs_root ip=dhcp console=ttyS1`

That uses DHCP to assign an IP address to the machine and puts its boot messages (console) on the second serial port. The following is the corresponding example using a static IP address; it also explicitly specifies Busybox's (non-standard) location for init:

- `root=/dev/nfs rw nfsroot=12.345.67.89:/path/on/server/to/nfs_root ip=12.345.67.88::12.345.67.1:255.255.255.0:embed-o-matic:eth0:off console=ttyS1 init=/bin/init`

5. Known problems

5.1 /sbin/init doesn't start.

A common problem with /sbin/init is that some distributions (e.g., Red Hat Linux) come with /sbin/init dynamically linked. So you have to provide a correct /lib setup to the client. An easy thing one could try is replacing /sbin/init (for the client) with a statically linked ``Hello World'' program. This way you know if it is something more basic, or ``just'' a problem with dynamic linking.

Also note that Busybox by default installs its `init` symlink in `/bin` rather than `/sbin`. You may need to move it or pass an explicit `init=` parameter on the kernel command line, as shown in the final example of the previous section.

5.2 /dev troubles.

If you get some garbled messages about ttys when booting, then you should run a MAKEDEV from the client in the /dev directory. There are rumors that this doesn't work with certain server OSes that use 64-bit device numbers; should you run into this, please consider updating this section! A potential solution would be to create a small /dev ram disk early in the boot process and reinstall the device nodes each time, or simply embed directly into the kernel a suitably initialized ramdisk.

6. Other resources

- In the Documentation directory of kernel source there is a file documenting NFS-Root systems (`Documentation/nfsroot.txt`).
- There are quite a few related HOWTOs:
    - Diskless-HOWTO (specifically, the *Network Booting* section)
    - Diskless-root-NFS-HOWTO
    - Diskless-root-NFS-other-HOWTO
    - Network-boot-HOWTO
    - PXE-Server-HOWTO ("Pre-boot eXecution Environment") < coming >

- There is a BOOTP client: `http://ibiblio.org/pub/Linux/system/network/admin/bootpc-0.64.tar.gz`

  With initrd (which is included in Linux 2.0), it could be made to work for diskless stations quite nicely. initrd is actually always an advanced option for more customized setups.
- For plain bootpd-based boots this is actually probably not needed as Linux 2.0 contains also the option to use BOOTP instead of RARP. (To be more precise, you can compile both in the kernel, and the faster response wins.)
- There is a patch floating around that allows for swapping over NFS. It was sent to me (during a private high workload phase), but I somehow managed to lose the mail.

  You can probably get it from http://www.linuxhq.com/ in the unofficial-patches section.