Root over nfs clients & server Howto.

Table of Contents

Root over nfs clients & server Howto.

Hans de Goede hans@highrise.nl

v1.0 30 March 1999

Howto setup a server and configure clients for diskless operation from a network.

5.Added bonus: booting from cdrom

6.Thanks

7.Comments

1.Introduction

This howto is also available at − http://xmame.retrogames.com/hans. This document describes a setup for nfs over root. This document differs from the other root over nfs howto's in 2 ways:

1. It describes both the server and the client side offering a complete solution, it doesn't desribe the generic principles off root over nfs although they will become clear. Instead it offers a working setup for root over nfs. One of the many possible setup's I might add.
2. This solution is unique in that it shares the root of the server with the workstations (ws). Instead of having a mini−root per ws. This has a number of advantages:
    - low diskspace usage
    - any changes on the serverside are also automagicly made at the client side, all configuration has only to be done once!
    - Very easy adding of new clients
    - only one system to maintain

This document is heavily based on a RedHat−5.2 system. Quite a bit of prior linux sysadmin experience is assumed in this howto, if you have that it shouldn't be a problem to addept this solutions to other distributions.

1.1 Copyright

partial copies.
* Any translation or derived work must be approved by the author in writing before distribution.
* If you distribute this work in part, instructions for obtaining the complete version of this manual must be included, and a means for obtaining a complete version provided.
* Small portions may be reproduced as illustrations for reviews or quotes in other works without this permission notice if proper citation is given.

Exceptions to these rules may be granted for academic purposes: Write to the author and ask. These restrictions are here to protect us as authors, not to restrict you as learners and educators.

1.2 Changelog

* v0.1, 20 January 1999: First draft written at the HHS, where the setup was originally developed.
* v1.0, 30 March 1999: First released version partially written in time of ISM

2.Basic principle

As already said with this setup the clients share basicly the entire root−fs with the server. But the clients ofcourse only get read access to it. This is basicly how things work.

2.1 Things can't be that simple

Unfortunatly things aren't that simple, there are a couple of problems the overcome with this simple setup.

Each ws needs its own writable copy of a number of dirs

A normal linux setup needs to have write access to the following dirs:

1. /dev
2. /var
3. /tmp

There are 3 solutions for this, of which one will only work for /dev:

1. mount a ramdisk and populate it by untarring a tarball, or by copying a template dir.
    * Advantages:
        1. It's cleaned up every reboot, which removes tmp files and logs. Thus it needs no maintaince unlike server sided dirs.
        2. It doesn't take up any space on the server, and that it doesn't generate any network traffic. A ramdisk takes less server and network resources, and is faster.
    * Disadvantages:

        1. It takes memory.
        2. The logs aren't kept after a reboot, if you really want logging of all your clients tell syslog to redirect the logging to your server.
2. create a dir for each ws on the server and mount it rw over nfs.
    - Advantages & disadvantages:
        1. The above arguments work in reverse for serversided dirs.
3. With kernel 2.2 devfs can be used for /dev, this is a virtual filesystem ala /proc for /dev.
    - Advantages:
        1. Devfs takes very little memory when compared to a ramdisk / no diskspace on the server and is very fast. A normal /dev takes at least 1.5 mb since the minimal size for a file (and thus for a device) is 1k, and there are somewhere around 1200 devices. You can offcourse use a template of a stripped /dev with only the entries you need to save some space. 1.5 Mb is a lott for a ramdisk and also isn't nice on a server.
        2. Devfs automagicly creates entries for newly added & detected devices, so no maintainance is needed.
    - Disadvantages:
        1. Any changes to /dev like creating symlinks for the mouse and cdrom are lost. Devfs comes with a script called rc.devfs to save these chances. The script's provided in this howto will automagicly restore these symlinks settings by calling rc.devfs If you make any changes to /dev you need to call the rc.devfs yourself to save them by typing:

           /etc/rc.d/rc.devfs save /etc/sysconfig

As you can see, there are a number of ways to solve this problem. For the rest of this Howto the following choices are assumed:

- For /dev we'll use Devfs
- For /var and /tmp we'll use a shared ramdisk of 1mb. It's shared to use the space as effeciently as possible. /tmp is replaced by a symlink to /var/tmp to make the sharing possible.
- Populating the ramdisk with tarballs or template dirs, works equally well. But with template dirs it's much easier to make changes, thus we'll use template dirs.

Write access to /home might be needed

Not really a problem in every unix client/server setup /home is mounted rw from the server so we'll just do that ;)

How does a ws find out it's ip so that it can communicate with the server?

Luckily for us, this problem has already been solved and the linux kernel has support for 2 ways of autoconfiguration of the ip−address:

1. RARP
2. Bootp

Rarp is the easiest to setup, bootp is the most flexible. Since most bootroms only support bootp that's what we'll use.

What about ws sepecific configuration

On redhat most system dependent config files are already in /etc/sysconfig We'll just move those which aren't there and add symlinks. Then we mount a seperate /etc/sysconfig on a per ws basis. This is really the only distribution dependent part on other distributions you can just create a sysconfig dir, move all config files which can't be shared there and create symlinks. Also /etc/rc.d/rc3.d, or symilar on other dists, might need to be different for the server resp the workstations. Assuming that all ws run the same services in runlevel 3, we'll just create a seperate 3th runlevel for the workstations and the server:

1. Create both a /etc/rc.d/rc3.ws and a /etc/rc.d/rc3.server
2. make /etc/rc.d/rc3.d a symlink to /etc/sysconfig/rc3.d
3. make /etc/sysconfig/rc3.d a symlink to the apropiate /etc/rc.d/rc3.xxx
4. replace S99local in rc3.ws by a link to /etc/sysconfig/rc.local so that each ws can have it's own rc.local

Miscelancious problems

There are a few problems left:

1. /etc/rc.d/rc.sysinit needs /var, so /var needs to be mounted or created before /etc/rc.d/rc.sysinit is run. It would also be nice if the ws−specific /etc/sysconfig is mounted before any initscripts are run.
   - We'll just source a bootup script for the ws in the very top of /etc/rc.d/rc.sysinit. Note this script will then ofcourse also be sourced by the server itself on boot, so the script has to detect this and do nothing on the server.
2. /etc/mtab needs to be writable:
   - This is a tricky one, just create a link to /proc/mounts and create an empty file mounts in /proc so that fsck and mount don't complain during the initscripts when /proc isn't mounted yet. One note smb(u)mount doesn't respect mtab being a link and overwrites it. Thus if you want to use smb(u)mount create wrapper scripts that restore the symlink.

3.Preparing the server

Now it's time to prepare the server to serve diskless clients.

3.1 Building a kernel

The first thing todo is build a kernel with the nescesarry stuff in to support root over nfs. Take the following steps to build your kernel:

1. Since we'll be using redhat−5.2 with kernel−2.2 you should asure yourself that your redhat−5.2 is kernel−2.2 ready. RedHat has got an excellent howto on this.
2. I use the same kernel for both server and ws, to avoid module conflicts since they share the same /lib/modules. If this is not possible in your situation, fake different kernel versions by editing the version number in the kernel's top makefile. These different versionsnumbers will avoid any conflicts.
3. Besides the usual stuff the kernel should have the following:
    - ext2 compiled in (if used on server, or for both)
    - nfs and root−over−nfs compiled in (if used on client or both), to get the nfs over root option in 2.2 enable ip−autoconfig in the network options. We'll use bootp as configuration method.
    - ws networkcard support compiled in (if used on client or both)
    - compile devfs in (required for client, also nice for server)
    - anything else you normally use, modules for all other devices used on either the server or all / some ws etc.
4. The kernel−src needs to be edited to make the default root−over−nfs mount: /tftpboot/<ip>/root instead of just /tftpboot/<ip>. This is to get a clean tree in /tftpboot with one dir per ws containing both the root for it (a link to the actual server root) and any ws specific dirs.
    - For 2.0 This is a define in: "include/linux/nfs_fs.h" called "NFS_ROOT"
    - For 2.2 This is a define in: "fs/nfs/nfsroot.c"
5. Now just compile the kernel as usual, see the kernel−howto.
6. If you don't have /dev/nfsroot yet, create it by typing:

    ```
    mknod /dev/nfsroot b 0 255.
    ```
7. After compiling the kernel set the root to nfsroot by typing:

    ```
    rdev <path−to−zImage>/zImage /dev/nfsroot
    ```
8. Before booting with devfs you need to make a few changes to /etc/conf.modules, append the contents of the conf.modules in the devfs documentation to it.
9. Since this new kernel is compiled for autoconfig of ip's it will try to autoconf the ip of the server during bootup. Which ofcourse will fail since it gives out the ip's. To avoid a long timeout add: append="ip=off" To the linux section of /etc/lilo.conf.
10. Run lilo and boot the new kernel.
11. Due to devfs you'll have lost all symlinks on the server. With redhat this is usually /dev/mouse and /dev/cdrom. Recreate these. If you also used to use special ownerships, chown to appropiate files in /dev. Now save the /dev settings (in /etc/sysconfig, since they might be ws specific):

- Copy rc.devfs from the devfs documentation in the kernel source to /etc/rc.d/rc.devfs and make it executable
- Save the settings by typing:

  /etc/rc.d/rc.devfs save /etc/sysconfig

3.2 Creating and populating /tftpboot, making symlinks for /tmp etc.

The next step is to create and populate /tftpboot

The automagic part

This is all handled by a big script since putting a long list of commands into this howto seemed pretty useless to me. If you want todo this manual just read the script and type it in as you go ;)

This setup script thus some nasty things like nuke /tmp, temporary kill syslog, umount /proc. So make sure that noone is using the machine during this, and that X isn't running. Just making sure your the only one logged in on a text−console is enough, no need to change runlevels.

DISCLAIMER: this script has been tested but nevertheless if it messes up your server your on your own. I can take no responsibility what so ever. Lett me repeat this howto is only for experienced linux sysadmins. Also this is script is designed to be run once and I really mean once. Running it twice will nuke: /etc/fstab, /etc/X11/XF86Config, /etc/X11/X and /etc/conf.modules.

Now with that said, just cut and paste the script make it executable, execute it and pray to the holy penguin that it works ;)

```
#!/bin/sh

SERVER_NAME=`hostname -s`

###
echo creating /etc/rc.d/rc.ws
#this basicly just echos the entire script ;)
echo "#root on nfs stuff

SERVER=$SERVER_NAME

#we need proc for mtab, route etc
mount -t proc /proc /proc

IP=\`ifconfig eth0|grep inet|cut --field 2 -d ':'|cut --field 1 -d ' '\`

#if the first mount fails we're probably the server, or atleast something is
#pretty wrong, so only do the other stuff if the first mount succeeds
mount \$SERVER:/tftpboot/\$IP/sysconfig /etc/sysconfig -o nolock &&
{
   #other mounts
   mount \$SERVER:/home /home -o nolock
   mount \$SERVER:/ /\$SERVER -o ro,nolock
```

```
    #/var
    echo Creating /var ...
    mke2fs -q -i 1024 /dev/ram1 1024
    mount /dev/ram1 /var -o defaults,rw
    cp -a /tftpboot/var /

    #network stuff
    . /etc/sysconfig/network
    HOSTNAME=\`cat /etc/hosts|grep \$IP|cut --field 2\`
    route add default gw \$GATEWAY
    ifup lo
}

#restore devfs settings
/etc/rc.d/rc.devfs restore /etc/sysconfig

umount /proc" > /etc/rc.d/rc.ws

###
echo splitting runlevel 3 for the client and server
mv /etc/rc.d/rc3.d /etc/rc.d/rc3.server
cp -a /etc/rc.d/rc3.server /etc/rc.d/rc3.ws
rm /etc/rc.d/rc3.ws/*network
rm /etc/rc.d/rc3.ws/*nfs
rm /etc/rc.d/rc3.ws/*nfsfs
rm /etc/rc.d/rc3.ws/S99local
ln -s /etc/sysconfig/rc.local /etc/rc.d/rc3.ws/S99local
ln -s /etc/rc.d/rc3.server /etc/sysconfig/rc3.d
ln -s /etc/sysconfig/rc3.d /etc/rc.d/rc3.d

###
echo making tmp a link to /var/tmp
rm -fR /tmp
ln -s var/tmp /tmp

###
echo moving various files around and create symlinks for them
echo mtab
/etc/rc.d/init.d/syslog stop
umount /proc
touch /proc/mounts
mount /proc
/etc/rc.d/init.d/syslog start
rm /etc/mtab
ln -s /proc/mounts /etc/mtab
echo fstab
mv /etc/fstab /etc/sysconfig
ln -s sysconfig/fstab /etc/fstab
echo X-config files
mkdir /etc/sysconfig/X11
mv /etc/X11/X /etc/sysconfig/X11
ln -s ../sysconfig/X11/X /etc/X11/X
mv /etc/X11/XF86Config /etc/sysconfig/X11
ln -s ../sysconfig/X11/XF86Config /etc/X11/XF86Config
echo conf.modules
mv /etc/conf.modules /etc/sysconfig
ln -s sysconfig/conf.modules /etc/conf.modules
echo isapnp.conf
mv /etc/isapnp.conf /etc/sysconfig
ln -s sysconfig/isapnp.conf /etc/isapnp.conf
```

```
###
echo creating a template dir for the ws directories
echo /tftpboot/template
mkdir /home/tftpboot
ln -s home/tftpboot /tftpboot
mkdir /tftpboot/template
mkdir /$SERVER_NAME
echo root
ln -s / /tftpboot/template/root
echo sysconfig
cp -a /etc/sysconfig /tftpboot/template/sysconfig
rm -fR /tftpboot/template/sysconfig/network-scripts
ln -s /$SERVER_NAME/etc/sysconfig/network-scripts \
 /tftpboot/template/sysconfig/network-scripts
echo NETWORKING=yes > /tftpboot/template/sysconfig/network
echo `grep "GATEWAY=" /etc/sysconfig/network` >> /tftpboot/template/sysconfig/network
echo "/dev/nfsroot / nfs defaults 1 1" > /tftpboot/template/sysconfig/fstab
echo "none /proc proc defaults 0 0" >> /tftpboot/template/sysconfig/fstab
echo "#!/bin/sh" > /tftpboot/template/sysconfig/rc.local
chmod 755 /tftpboot/template/sysconfig/rc.local
rm /tftpboot/template/sysconfig/rc3.d
ln -s /etc/rc.d/rc3.ws /tftpboot/template/sysconfig/rc3.d
rm /tftpboot/template/sysconfig/isapnp.conf
echo var
cp -a /var /tftpboot/var
rm -fR /tftpboot/var/lib
ln -s /$SERVER_NAME/var/lib /tftpboot/var/lib
rm -fR /tftpboot/var/catman
ln -s /$SERVER_NAME/var/catman /tftpboot/var/catman
rm -fR /tftpboot/var/log/httpd
rm -f /tftpboot/var/log/samba/*
for i in `find /tftpboot/var/log -type f`; do cat /dev/null > $i; done
rm `find /tftpboot/var/lock -type f`
rm `find /tftpboot/var/run -type f`
echo /sbin/fsck.nfs
echo "#!/bin/sh
exit 0" > /sbin/fsck.nfs
chmod 755 /sbin/fsck.nfs

echo all done
```

Manual adjustments to some files

Now we need to make a few manual adjustments to the server:

1. The ws setup script has to be sourced at the very beginning of rc.sysinit, so add the following lines directly after setting the PATH:

```
#for root over nfs workstations.
/etc/rc.d/rc.ws
```

2. Strip /etc/rc.d/rc3.ws to a bare minimum. It might be useful to create something like rc.local.ws but I'll leave that up to you. Network and nfsfs are already setup.The following have been already

removed / updated by the automagic script:

- network
- nfsfs
- nfs
- rc.local

3.3 Exporting the appropriate file systems and setting up bootp

The server must ofcourse export the appropriate filesystems and asign the ip addresses to the clients.

Exporting the appropriate file systems

We need to export some dir's for the workstations so for the situation here at the university I would add the following to /etc/exports:

```
/ *.st.hhs.nl(ro,no_root_squash)
/home *.st.hhs.nl(rw,no_root_squash)
```

Ofcourse use the apropriate domain ;) and restart nfs by typing:

/etc/rc.d/init.d/nfs restart

Note for knfsd users: knfsd doesn't allow you to have multiple exports on one partition with different permissions. Also knfsd doesn't allow clients to go past partition boundaries for example if a client mounts / and /usr is a different partition it won't have access to /usr. Thus if you use knfsd, at least /home should be on a different partition, the server prepare script already puts /tftpboot in /home so that doesn't need a seperate partition. If you've got any other partitions your clients should have access to export them seperatly and add mount commands for them to /etc/rc.d/rc.ws.

Setting up bootp

1. If bootp isn't installed yet install it. It comes with RedHat.
2. Edit /etc/inetd.conf and uncomment the line beginning with bootps, if you want to use a bootprom uncomment tftp while your at it.
3. Restart inetd by typing:

   /etc/rc.d/init.d/inetd restart

4.Adding workstations

Now that the server is all done, we can start adding workstations.

4.1 Creating a boot disk or bootrom

You'll need ot create a bootrom and / or a bootdisk to boot your workstation.

Creating a bootdisk

Even if you wish to use a bootrom its wise to first test with a bootdisk, to create a boot disk just type:

```
dd if=/<path−to−zImage>/zImage of=/dev/fd0
```

Creating a bootrom

There are a few free package's out there to create bootroms:

1. netboot, this is IMHO the most complete free package out there. It uses standard dos packet drivers so allmost all cards are supported. One very usefull hint I got on there mailing list was to pklite the packetdrivers since some commercial drivers are to big to fit into the bootrom. Netboot's documentation is complete enough, so I won't waste any time reproducing it here, it should be more then sufficient to create a bootrom and boot a ws with it. Netboot's webpage is: http://www.han.de/~gero/netboot/
2. etherboot, this is the other free package out there it has got a few nice features like dhcp support, but has limited driver support as it uses its own driver format. I haven't used this so I really can't give anymore usefull info. Etherboot's webpage is: http://www.slug.org.au/etherboot/

About the roms themselves. Most cards take ordinary eproms with an 28 pins dip housing. These eproms come in size upto 64kB. For most cards you'll need 32kB eproms with netboot. Some cards drivers will fit into 16kB but the price difference of the eproms is minimal. These eproms can be burned with any ordinairy eprom burner.

4.2 Creating a ws dir

Just copy over the template by typing:

```
cd /tftpbootcp −a template <ip>
```

You could of course also copy over the dir of a workstation with identical mouse, graphicscard and monitor and ommit the configuration in step 5.4.

4.3 Add entries to /etc/bootptab and /etc/hosts

Edit /etc/bootptab and add an entry for your test ws, an example entry is:

```
nfsroot1:hd=/tftpboot:vm=auto:ip=10.0.0.237:\
:ht=ethernet:ha=00201889EE78:\
:bf=bootImage:rp=/tftpboot/10.0.0.237/root
```

Replace nfsroot1 by the hostname you want your ws to have. Replace 10.0.0.237 by the ip you want your ws to have (do this twice) and replace 00201889EE78 by the MAC−ADDRESS of your ws. If you don't know the MAC−ADDRESS of the ws, just boot it with the just created boot disk and look for the MAC−ADDRESS in the boot messages. There's a chance bootpd is already running so just to make sure try to restart it by typing:

> killall −HUP bootpd

Don't worry if it fails, that just means it wasn't running, inetd will start it when asked too.

4.4 Booting the ws for the first time

Just boot the ws from the bootdisk. This should get you a working ws in textmode, with the exact same setup as your server except for the ip−nr and the running services. Even if you want to use a bootprom it's wise to first test with the bootdisk, if that works you can try to boot with the bootrom see the bootroms documentation for more info.

4.5 Set the ws specific configuration.

Now it's time to configure any ws specific settings:

1. First off all to get the mouse working, just run mouseconfig. To apply the changes, and check that the mouse works type:

   > /etc/rc.d/init.d restart

2. Run Xconfigurator, when Xconfigurator has probed the card and you can press ok don't! Since we have moved the symlink for the Xserver from /etc/X11/X to /etc/sysconfig/X11/X Xconfigurator will fail to create the proper link. Thus to make sure the rest of Xconfigurator goes well, switch to another console and create the link in /etc/sysconfig/X11 to the advised server. Now just finish Xconfigurator and test X.
3. Configure anything else which is different then the server / template:

   ♦ sound: You probaly need to modify isapnp.conf and conf.modules, both are already made links to /etc/sysconfig by the server setup script.

    - ♦ cdrom: Link in /dev, entry in /etc/fstab? etc.
    - ♦ rc.local: Make any nescesarry changes.
4. Save the links and any other changes to /dev type:

   /etc/rc.d/rc.devfs save /etc/sysconfig

5. All done.

5.Added bonus: booting from cdrom

Much of the above also goes for booting from cdrom. Since I wanted to document howto boot from cdrom anyway, I document it in here to avoid typing a lott of the same twice.

Why would one want to boot a machine from cd−rom? Booting from cdrom is interesting everywhere where one wants to run a very specific application, like a kiosk, a library database program or an intenet cafe, and one doesn't have a network or a server to use a root over nfs setup.

5.1 Basic Principle

The basic principle is wants again simple, boot with a cdrom as root. To make this possible we'll use the rockridge extension to put a unix like filesystem on a cd and the Eltorito extension to make cd's bootable.

Things can't be that simple

Ofcourse this setup also has a few problems. most are the same as above:

1. We'll need write access to: /dev, /var & /tmp.
    - ♦ We'll just use the same solutions as with root over nfs (see above):
        - ◊ For /dev we'll use Devfs
        - ◊ For /var and /tmp we'll use a shared ramdisk of 1mb. It's shared to use the space as effeciently as possible. /tmp is replaced by a symlink to /var/tmp to make the sharing possible.
        - ◊ Populating the ramdisk with tarballs or template dirs, works equally well. But with template dirs it's much easier to make changes, thus we'll use template dirs.
2. Some apps need write access to /home.
    - ♦ Put the homedir of the user's who will be running the application in /var, and populate it wiht the rest of /var every boot.
3. /etc/mtab needs to be writable:
    - ♦ Create a link to /proc/mounts and create an empty file mounts in /proc, see above.

5.2 Creating a test setup.

Now that we know what we want todo and how, it's time to create a test setup:

1. For starters just take one of the machines which you want to use and put in a big disk and a cd−burner.
2. Install your linux of choice on this machine, and leave a 650mb partition free for the test setup. This install will be used to make the iso−image and to burn the cd's from, so install the nescesarry tools. It will also be used to restore any booboo's which leave the test setup unbootable.
3. On the 650 mb partition install your linux of choice with the setup you want to have on the cd, this will be the test setup
4. Boot the test setup.
5. Compile a kernel as described in Section 3.1, follow all the steps, the changes need for devfs are still needed! At step 3 of Section 3.1 put in the following:
    - isofs compiled in
    - devfs compiled in
    - cdrom support compiled in
    - everything else you need either compiled in or as module.
6. Configure the test setup:
    - Create the user which we'll be running the application.
    - Put it's homedir in /var.
    - Install the application if needed.
    - Configure the application if needed.
    - Configure the user so that the application is automagicly run after login.
    - Configure linux so that it automaigcly logs in the user.
    - Configure anything else which needs configuring.
7. Test that the test setup automagicly boots into the apllication and everything works.
8. Boot the main install and mount the 650 mb partition on /test of the main install.
9. Put the following in a file called /test/etc/rc.d/rc.iso, this file we'll be sourced at the begining of rc.sysinit to create /var

```
#/var
echo Creating /var ...
mke2fs -q -i 1024 /dev/ram1 1024
mount /dev/ram1 /var -o defaults,rw
cp -a /lib/var /

#restore devfs settings, needs proc
mount -t proc /proc /proc
/etc/rc.d/rc.devfs restore /etc/sysconfig
umount /proc
```

10. Edit /test/etc/rc.sysinit comment the lines we're the root is remounted rw and add the following 2 lines directly afer setting the PATH:

```
#to boot from cdrom
```

```
. /etc/rc.d/rc.iso
```

11. Copying the following to a script and executing it, this wil create a template for /var and make /tmp and /etc/mtab links.

```
#!/bin/sh
echo tmp
rm -fR /test/tmp
ln -s var/tmp /test/tmp

###
echo mtab
touch /test/proc/mounts
rm /test/etc/mtab
ln -s /proc/mounts /test/etc/mtab

###
echo var
mv /test/var/lib /test/lib/var-lib
mv /test/var /test/lib
mkdir /test/var
ln -s /lib/var-lib /test/lib/var/lib
rm -fR /test/lib/var/catman
rm -fR /test/lib/var/log/httpd
rm -f /test/lib/var/log/samba/*
for i in `find /test/lib/var/log -type f`; do cat /dev/null > $i; done
rm `find /test/lib/var/lock -type f`
rm `find /test/lib/var/run -type f`
```

12. Remove the creation of /etc/issue* from /test/etc/rc.local it will only fail.
13. Now boot the test partition again, it will be read only just like a cdrom. If something doesn't work reboot to the working partition fix it, try again etc. Or you could remount / rw ,fix it then reboot straight into to test partition again. To remount / rw type:

    mount –o remount,rw /

5.3 Creating the cd

Creating a boot image

First of all boot into the workign partition. To create a bootable cd we'll need an image of a bootable floppy. Just dd–ing a zimage doesn't work since the loader at the beginning of the zimage doesn't seem to like the fake floppydrive a bootable cd creates. So we'll use syslinux instead.

1. Get boot.img from a redhat cd
2. Mount boot.img somewhere through loopback by typing:

mount boot.img somewhere −o loop −t vfat

3. Remove everything from boot.img except for:
   - ldlinux.sys
   - syslinux.cfg
4. Cp the kernel−image from the test partition to boot.img.
5. Edit syslinux.cfg so that it contains the following, ofcourse replace zImage by the appropiote image name:

```
default linux

label linux
kernel zImage
append root=/dev/<insert your cdrom device here>
```

6. Umount boot.img:

   umount somewhere

7. If your /etc/mtab is a link to /proc/mounts umount won't automagicly free /dev/loop0 so free it by typing:

   losetup −d /dev/loop0

Creating the iso image

Now that we have the boot image and an install that can boot from a readonly mount it's time to create an iso image of the cd:

1. Copy boot.img to /test
2. Cd to the directory where you want to store the image make sure it's on a partition with enough free space.
3. Now generate the image by typing:

   mkisofs −R −b boot.img −c boot.catalog −o boot.iso /test

Verifying the iso image

1. Mounting the image throug the loopbackdevice by typing:

   mount boot.iso somewhere −o loop −t iso9660

2. Now verify that the contents is ok.

3. Umount boot.iso:

   umount somewhere

4. If your /etc/mtab is a link to /proc/mounts umount won't automagicly free /dev/loop0 so free it by typing:

   losetup −d /dev/loop0

Writing the actual cd

Assuming that you've got cdrecord installed and configured for your cd−writer type:

```
cdrecord −v speed=<desired writing speed> dev=<path to your writers generic scsi device>
boot.iso
```

5.4 Boot the cd and test it

Well the title of this paragraph says it all ;)

6.Thanks

- The HHS (Haagse Hoge School) a dutch college where I first developed and tested this setup for use in a couple of labs. And where the initial version of this HOWTO was written.
- ISM a dutch company where I'm doing my final project. Part of the project involves diskless machines, so I got to develop this setup further and had the time to update this HOWTO.
- All the users who will give me usefull input once this first version is out ;)

7.Comments

Comments suggestions and such are welcome. They can be send to Hans de Goede at: j.w.r.degoede@et.tudelft.nl

---