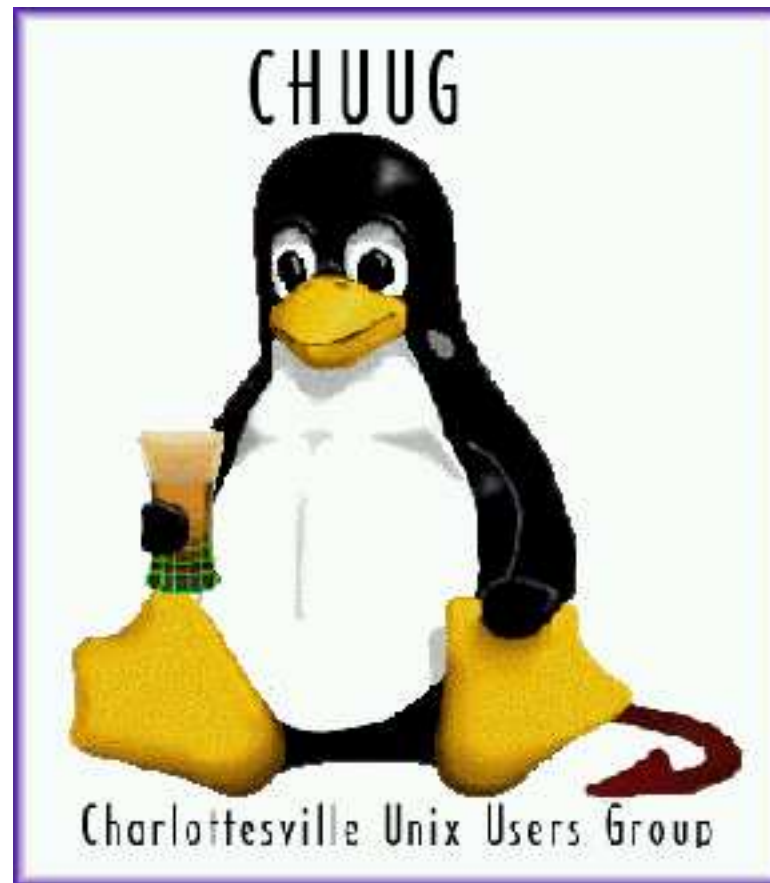


# A better way to install that OS

**Rapid deployment and diskless workstations  
using net-boot technology**



**Joshua Malone** ([jmalone@ubergeeks.com](mailto:jmalone@ubergeeks.com))

# What is net-booting?

---

## The Basics:

- System boots from BIOS on the ethernet card
  - Built-in to modern NICs
  - Can add a boot ROM to older NICs
- Requests kernel and other files from a boot server
- No disks (of any kind) required

## Benefits:

- No need to make boot disks
- Repeatable (when combined with automated installation)
- Boot as many machines simultaneously as you want
  - Limited by network (instead of boot media)

# What is automated installation?

---

Installer gets its configuration from a file instead of a human

- Quicker (no need to type anything)
- Less error-prone (no possibility of mis-typing anything)
- Repeatable and self-documented

We'll focus on RedHat's "kickstart" and FreeBSD's "install.cfg"

# Rapid deployment vs. diskless booting

---

The goal of rapid deployment is to net-boot the system and install an operating system to a local disk with minimal user-intervention.

The goal of diskless booting is to create centrally-managed workstations with little or no local information.

Net-booting can help us achieve both of these goals by reducing boot time, eliminating the need for boot media and automating start-up.

# How normal booting works (a quick background)

---

Boot stages:

- Stage 0: POST and BIOS
- Stage 1: Master boot record / boot block
- Stage 2: Operating system loader
- Stage 3: OS kernel

Stage 0 is the motherboard's BIOS (plus other cards like SCSI, NIC, etc.)

Stage 1 loader is 512-byte block on hard drive.

- Knows only enough to find stage 2 loader and run it

Stage 2 loader is native to the OS

- Usually located at the beginning of the partition
- Knows how to find kernel, get boot options, find root FS, etc.

Get gradually more complex until the kernel is actually running.

Lack of space in the first stage necessitates this "chaining" of boot loaders.

# How net-booting works

---

Process:

- POST, BIOS, boot ROM
- System makes 'bootp' request (broadcast) ---->
- <---- Bootp server responds with IP info plus boot filename
- System loads boot file via TFTP (trivial file transfer protocol)
  - Boot file is usually a second-stage bootloader for the OS
- Boot ROM executes boot file to load the real OS
- Second-stage bootloader loads the rest of the OS via tftp or nfs
  - Kernel
  - Root filesystem image
  - Possibly kernel modules
- Second-stage bootloader executes OS (kernel) and OS takes over from there

# Second-stage bootloader

---

Linux usually uses 'LILO' or 'GRUB' for its second-stage bootloader, which takes care of loading the kernel (and possibly root filesystem image) and passing arguments (i.e., video mode, driver options, etc.)

FreeBSD uses 'loader' to load kernel, modules, root filesystem, etc.

These bootloaders know how to access the disks (BIOS calls) and filesystems to load these files.

For net-booting, we need a second-stage loader than understands ethernet and a network protocol instead of disks and filesystems.

# Second-stage bootloader: FreeBSD

---

pxeboot - A PXE stub attached to 'loader' for use in net-booting

Can load kernel, rootfs image, modules and config files just like booting from local disk

pxeboot receives options for boot file from bootp server:

- 'next-server' - IP of server to get root FS from
- 'root-path' - NFS filesystem to mount as root (if using NFS)

See man(8) pxeboot

# Second-stage bootloader: Linux

---

pxelinux - A version of syslinux for use in net-booting

(There are also other variants of syslinux for making bootable CDs and floppies)

NOTE: pxelinux can also boot a special "kernel" called 'memdisk' which can boot any bootable floppy image over net-boot

*<http://syslinux.zytor.com/pxe.php>*

# Second-stage bootloader: bpbatch

---

Bpbatch is a sophisticated net-boot loader that can

- Boot many operating systems, including:
  - Linux
  - FreeBSD
  - DOS
  - Even Windows!
- Restore a disk image to a local hard disk
- Authenticate users using LDAP or RADIUS

Provides more "eye-candy" than the others

*<http://www.bpbatch.org/>*

# What is PXE?

---

PXE is the Preboot eXecution Environment

Provides:

- First-stage bootloader
- NDIS ethernet drivers for second-stage loader
  - Device-agnostic ethernet drivers

# Using pxeboot

---

Pxeboot is just like loader!

- Same syntax as loader
- Reads /boot/loader.rc to start
- Starts with `currdev=pxe0`

Loads files via NFS or TFTP

- Must set `LOADER_TFTP_SUPPORT="YES"` for loader to use TFTP
  - Either in 'make' command or in /etc/make.conf for buildworld
- Run '`make -DLOADER_TFTP_SUPPORT="YES"`' in /usr/src/sys/boot/i386/ to build 'pxeboot' in pxeldr/

Syntax is exactly the same as loader

Can prompt for interactive information at boot time

- Choose kernel
- Change boot media
- Choose initial root filesystem

# Using pxeboot (continued)

---

Since pxeboot is just loader, it can boot an installed OS, too!

- Set `CURRDEV=disk0s1a`
- Set `vfs.root.mountfrom="/dev/ad0s1a"`

Serial console support, too!

- Set `console="comconsole"`
- Switches interactive input and output to serial port

Load kernel configuration file:

- `load -t userconfig_script /boot/kernel.conf`
- If this file is empty, the kernel will bypass the configuration step.

See `/boot/defaults/loader.conf` for more options

# Example: Using pxeboot to choose kernel and mfsroot interactively

---

/boot/loader.rc

```
echo "PXEBoot interactive menu"
read -p "Choose kernel file: " kernel
load /$kernel
read -p "Choose mfsroot file image: " mfsroot
load -t mfs_root /$mfsroot
echo "Booting the system, stand by...."
autoboot 5
```

A forth interpreter is also available in pxeboot/loader

```
▫include /boot/loader.4th
```

# Using pxelinux

---

Pxelinux looks for config files in directory /pxelinux.cfg/

Configuration is same as 'syslinux' except for config file name

Looks for config file in "old-school" net-boot fashion:

- Try file named by IP address in hex (all caps); eg: 0A000017
- If not found, strip off least-significant hex digit and try again
- Finally, try file 'default'

Important Note: pxelinux requires a TFTP server that supports the 'tsize' option. Supported servers include:

- tftp-hpa (<http://www.kernel.org/pub/software/network/tftp>)
- atftp (<ftp://ftp.mamalinux.com/pub/atftp>)
- tftpd32 (Win32) (<http://tftpd32.jounin.net>)

(FreeBSD's stock tftpd does NOT support this)

# Pxelinux: configuration syntax

---

- ▣ `DEFAULT <kernel> <options>`
  - Sets default boot command, kernel, kernel options
- ▣ `DISPLAY <filename>`
  - Displays the named file - useful for instructions
- ▣ `LABEL <labelnum>`
  - Define a boot configuration that can be selected
- ▣ `PROMPT [0|1]`
  - 0 - only display boot prompt if shift or alt is pressed
  - 1 - always display boot prompt
- ▣ `TIMEOUT <time>`
  - Timeout before running 'DEFAULT' (time is in 1/10 sec)
- ▣ `SERIAL <portnum> <speed>`
  - Place console on serial port (also stays on CRT)

# Pxlinux: configuration syntax (continued)

---

Options inside the 'LABEL' definitions

- `KERNEL <image>`
  - Choose kernel image
- `APPEND <options>`
  - Supply kernel command line options
- `LOCALBOOT 0`
  - Boot from local disk instead of network

Note: This is different from LILO's syntax

# Example: Using pxelinux

---

Simple configuration that tells pxelinux to boot from local disk if user doesn't select an image in 20 seconds. Image 2 uses the 'memdisk' kernel to load and boot a floppy disk image. Also outputs on serial port for headless machine

```
SERIAL 0 9600
DEFAULT 3
TIMEOUT 200
DISPLAY bootmsg.txt
PROMPT 1
LABEL 1
    KERNEL vmlinuz
    APPEND initrd=initrd.gz
LABEL 2
    KERNEL memdisk
    APPEND initrd=utildisk.img
LABEL 3
    LOCALBOOT 0
```

# Using bpbatch

---

Set command-line for bpbatch

- DHCP option 155 for PXE 2.0
- DHCP option 135 for PXE 1.1
- Use '-i' for interactive mode
- Use '<scriptname>' to run a script <scriptname>.bpb

Ex: specify

```
option option-155 "-i"
```

in 'dhcpd.conf' to boot interactively

See <http://cui.unige.ch/info/pc/remote-boot/howto.html> for more info

See <http://www.isc.org/ml-archives/dhcp-server/2000/06/msg00226.html> for info on Intel Boot Agents not requesting option-155.

(Note: I couldn't get isc-dhcpd3.0.1rc11 to take option-155)

# Net-booting FreeBSD

---

## Requirements:

- kernel
- root filesystem (MFS image or NFS export)
- pxeboot loader

## Kernel must have:

- `options MFS`
- `options MD_ROOT`
- `options NFS` (if using NFS instead of MFS)
- `options NFS_ROOT` (if using NFS instead of MFS)
- drivers for your network card

In most cases, **GENERIC** (the kernel from floppy or CD) will do just fine.

(Unless you need a non-standard NIC or disk driver -- may be able to load as modules, though)

# Net-booting Red Hat Linux

---

## Requirements:

- kernel (vmlinuz)
- root filesystem (initrd image or NFS export)
- pxelinux

## Kernel must have:

- "Initial ramdisk support" (if using initrd)
- "NFS filesystem support" (if using NFS)
- "Root filesystem on NFS" (if using NFS)
- "BOOTP support"
- "Kernel level autoconfiguration" (for 2.2 kernels)
- drivers for your NIC

Red Hat provides a kernel and initrd designed specifically for PXE/net-booting:

See mirrors and look under `/redhat/7.1/i386/images/pxeboot`

# NFS or TFTP/MFS?

---

You can get your root filesystem one of two ways:

- Mount an NFS export from another server
- Load an initial filesystem image (initrd/mfsroot)

NFS allows you to mount a large filesystem that would be impractical to pull over the network and load completely into memory

- Useful for a diskless workstation
- Complicated for simple automated install
- Every system shares the same filesystem contents

TFTP pulls a small filesystem image over the network and loads it into memory

- Useful for loading a small filesystem for simple tasks
- Doesn't require NFS server or NFS support in kernel (smaller kernel)
- Each system has its own copy in memory
- Must generate a filesystem image
- Doesn't require a nameserver (NFS gets pissy without one)

TFTP is best choice for automated installs if you're always installing the same (or similar) OS

NFS is best choice for a diskless workstation or when prototyping an FS image

# Making net-boot more useful: Custom root filesystem image

---

Since we're not booting from floppy, we can make a much larger initial filesystem image with more utilities.

FreeBSD: use `'vnconfig'` to create a UFS filesystem image

▫ use `'mdconfig'` for FreeBSD 5.x

Linux: use loop-back mounting to create an ext2 filesystem image

Also, if we're willing (or opting) to use NFS, we can NFS-mount a very complete initial filesystem.

# Making net-boot more useful: Custom root filesystem image

---

Example: making custom mfsroot on FreeBSD

```
dd if=/dev/zero of=mfsroot bs=1m count=<size>
vnconfig -s labels vn0 mfsroot
disklabel -r -w vn0
newfs /dev/vn0c
vnconfig -u vn0
vnconfig -r labels -c vn0 mfsroot
mount /dev/vn0 /mountpoint
```

Copy whatever utilites you want into the new filesystem.

```
umount /mountpoint
vnconfig -u vn0
```

Viola! We have our custom initial filesystem (mfsroot)

(The unconfig-config sequence is needed to clear the 'labels' flag)

# Making net-boot more useful: Custom root filesystem image

---

Some things to remember:

- Begin with the distribution boot/root floppy
- Rescue or 'fixit' floppy is a good place to begin adding things
- /bin and /sbin are statically linked in FreeBSD (changing in 5.x)
- Hard links are your friends :)

If you want to make your own "crunched" binary for mfsroot, check out 'crunchgen' utility.

# Setting up bootp and tftp for net-booting

---

PXE needs a bootp server to configure its network and get boot parameters.

Once bootp is done, PXE needs a tftp server to download the boot file from.

ISC-DHCPD will server bootp as well as dhcp:

- Set option 'allow bootp' to respond to bootp requests

# Playing nicely with another DHCP server

---

If you're forced to net-boot on a LAN with an existing DHCP server, there is a way to make sure you play nicely and

- Don't give addresses to non net-booting clients
- Don't accept offers from other DHCP servers

You can set up a "private" DHCP server with ISC-DHCPD by using a VCI

(vendor class identifier)

# Playing nicely with another DHCP server

---

In 'dhcpd.conf':

- If using PXE, specify

```
class "netbooters" {  
    match if substring (option vendor-class-identifier, 0, 9) = "PXEClient";  
    <more options> }
```

- If using Etherboot, specify

```
class "netbooters" {  
    match if substring (option vendor-class-identifier, 0, 9) = "Etherboot";  
    <more options> }
```

- Then, allocate a pool of addresses specifically for net-boot clients:

```
pool {  
    allow members of "netbooters";  
    range dynamic-bootp <ip-start> <ip-end>; }
```

Notes:

- 'class' declaration should occur outside a 'subnet' declaration
- 'pool' declaration should occur inside a 'subnet' declaration

# Setting up FTP server

---

Most installers can fetch install media via anonymous FTP ("FTP install")

## FreeBSD:

- Sysinstall expects to find an FTP tree like: `/pub/FreeBSD/releases/i386/4.4-RELEASE/`
- You can literally copy the CD to (or mount the CD under) this path.
- Neat trick: Download the ISO CD image and `vnconfig/mount` (or `loopback-mount` under Linux) it at this path

## Red Hat:

- Kickstart is much more flexible about accepting an FTP install path
- Use option `'url'` to tell kickstart the location of the install media
  - Can be `http://` or `ftp://`
- Examples:
  - `url --url ftp://<user>:<pass>@<server>/<path>`
  - `url --url ftp://redhat.cac.washington.edu/redhat/linux/7.1/en/os/i386/`
- Expects tree below `'url'`:

```
i386/  
  RedHat/  
    RPMS/  
      base/
```

I think everything below `i386/` looks like the CD, but I last used Red Hat 6.2

# Enabling anonymous FTP under unix

---

To turn on anonymous FTP under most unix-like operating systems, edit `/etc/inetd.conf` and uncomment/add line:

```
ftp      stream  tcp      nowait  root    /usr/libexec/ftpd      ftpd -l -A
```

FTP root directory should be the home directory of the user 'ftp'

Under FreeBSD, you can enable anonymous FTP using 'sysinstall':

- Configure -> Networking -> Anon FTP
- Must `mkdir` the directory beforehand

Under Red Hat, you can enable anonymous FTP using 'serviceconf':

- FTP is an xinetd service

Make sure 'inetd' (or 'xinetd') is running!

# Automated install: FreeBSD

---

FreeBSD uses an install utility called 'sysinstall' (located in /stand). Sysinstall will look for a configuration file called 'install.cfg' in / on the initial filesystem. If present, sysinstall will begin performing the installation immediately.

(Note: 'sysinstall' moving on FreeBSD 5.x)

There is no warning when using install.cfg - your disk WILL BE OVERWRITTEN

Basic tasks:

- Partition disk
- Lay out filesystems
- Configure networking for FTP/NFS install
- Choose distributions to install
- Install packages

There is a trick you can use to do just about anything during the scripted install; this is a great place to perform post-install tweaks or do things more advanced than sysinstall's basic functionality.

# **Sysinstall: 'install.cfg' syntax**

---

Warning: sysinstall is a little bit broken (read brain-dead)

I'll try to cover things that I know are broken/buggy/differ from the manpage.

There are LOTS of undocumented variables/functions (versus man (8) sysinstall)

Use the source, Luke!

# Sysinstall: 'install.cfg' syntax

---

## Basic idea:

- Set a variable using 'varname=value'
  - Used for setting function options
- Call a function using 'functionname'
- If a function expects options, you must set them before calling the function

## Partitioning disks:

- Function 'diskPartitionEditor' creates 'fdisk'-style slices
- Function 'diskLabelEditor' creates 'disklabel'-style partitions
- Function 'diskPartitionWrite' writes out slices/fdisk partitions
- Function 'diskLabelCommit' writes disklabel partitions

## Variables:

disk=<disk device name> - eg: ad0,da0

partition=<partition scheme> - eg: all,exclusive,existing

• Note: 'somenumber' is pretty broken

bootManager=<type> - eg: boot,standard,none

diskInteractive - run interactive disk partition editor

• Kind of defeats the purpose, but great for debugging

# install.cfg: Setting partitions

---

Format: <diskname>s<slice\_num>-<partition\_num>=<fs\_type> <size>  
<mountpoint> <flags>

Where:

- diskname is ad0,da1,etc.
- slice\_num is 1,2,3,4 ('fdisk' partition number)
- partition\_num is 1..7
  - Sysinstall will assign partition letters based on convention
  - Why not just use letters?!? Don't ask...
- size is FS size in 512-byte blocks
  - use '0' for "the rest of the disk"
- mountpoint is the FS's mount point
- flags are 1=enable soft-updates or leave empty

# install.cfg: Setting partitions (continued)

---

Examples:

```
ad0s1-1=ufs 262144 /
```

Creates a 128 MB ufs filesystem for / (the root FS)

```
ad0s1-2=swap 2097152
```

Creates a 1 GB swap partition

```
ad0s1-3=ufs 4194304 /usr 1
```

Creates a 2 GB /usr filesystem with soft-updates enabled

# install.cfg: Networking

---

## Functions:

- `mediaSetFTP` - configures networking for FTP install and chooses server
- `mediaSetNFS` - configures networking for NFS install

## Variables:

`hostname=<name>` - set system's hostname (during install only)

`nameserver=<IP>` - IP address of nameserver

`netDev=<devicename>` - networking device to use; eg: `fxp0,xl1`

`tryDHCP=<YES|NO>` - use DHCP for networking? **BUGGY**

`ipaddr=<IP>` - IP address to use for FTP install; eg: `10.0.0.20`

`netmask=<mask>` - netmask for interface; eg: `255.255.255.0, 0xffffffff`

`nfs=<path>` - full hostname:path of NFS export containing installation media

`_ftpPath=<path>` - full `ftp://<hostname>/<path>` to installation media

# install.cfg: Committing installation

---

## Functions:

- `installCommit` - Installs the selected dists and commits all changes to disk
- `installFilesystems` - Just writes filesystems to the disk (doesn't perform full installation)

**Note:** putting both `installFilesystems` and `installCommit` in the config file will confuse `sysinstall` and installation will fail.

- `shutdown` - Reboots the system

# install.cfg: Example

---

```
disk=ad0
partition=all
bootManager=standard
diskPartitionEditor
ad0s1-1=ufs 2621444 /
ad0s1-2=swap 2097152 none
ad0s1-3=ufs 262144 /var 1
ad0s1-4=ufs 262144 /tmp 1
ad0s1-5=ufs 4194304 /usr 1
ad0s1-6=ufs 0 /home 1
diskLabelEditor
diskPartitionWrite
installFilesystems
hostname=mybox.mydomain
netDev=xl0
tryDHCP=YES
_ftpPath=ftp://ftp14.freebsd.org/pub/FreeBSD/
mediaSetFTP
distSetXUser
installCommit
shutdown
```

# Automated install: Red Hat Linux

---

Red Hat has an install mode called "kickstart" which is scriptable. Kickstart looks for a file called 'ks.cfg' on either root filesystem or a DOS-formatted floppy disk.

You can specify the kickstart config file using DHCP:

```
filename "/ks.cfg";  
next-server your.nfs.server;
```

but this will keep you from being able to specify a netboot file. If you are netbooting, it may just be easier to put the file in the initrd or on NFS root.

Kickstart can script just about every element of the installer!

- Disk partition, network setup
- Install specific applications (it takes a list of RPMs)
- Execute shell commands (great for post-install tweaks)
- Configure settings of the installed system (without editing conf files)

Red Hat provides a graphical "kickstart" configuration editor:

```
/usr/sbin/ksconfig
```

# Red Hat Linux: Kickstart

---

Kickstart is well documented in Red Hat's online docs:

*<http://www.redhat.com/docs/manuals/linux/RHL-7.3-Manual/custom-guide/s1-kickstart2-options.html>*

Kickstart has evolved throughout Red Hat's releases, so be sure to check docs for your version of Red Hat.

To start a kickstart install use boot command-line option `ks=<mode>`

- For file on initrd, use `ks=file: /<filename>`
- For file on nfs server, use `ks=nfs:<server>: /<path>`

Note: These options are kernel command line options - pass them using 'APPEND' line in LILO or pxelinux

# Post-install: FreeBSD

---

Option 1: Make a custom post-install package

Use 'pkg\_create' to generate a custom package with an install script that performs your desired tweaks.

Script 'pre' is run before installing the package; 'post' is run after installing.

Alfred Perlstein has written a tool to help generate custom packages for just this purpose

*<http://people.freebsd.org/~alfred/pxe/pkgmaker.sh>*

See *<http://people.freebsd.org/~alfred/pxe/>* for more info on custom packages.

# Post-install: FreeBSD (continued)

---

Option 2: Use `system` function in 'install.cfg' to run an arbitrary command

Variable `command` is the name of the command to execute.

Major drawbacks:

- Difficulty seeing output of the command
- Command runs from 'system(3)' instead of a shell proper
- There can be only one! (But it can be a long command)

# Post-install: FreeBSD (continued)

---

Option 3: Use a trick to run a shell script/binary of your choosing

Sysinstall can execute an 'Emergency Holographic Shell' with the command

```
installFixitHoloShell
```

This causes sysinstall to exec /bin/sh in the root of the filesystem....

Soooooo....if there just happened to be a /.profile in our custom root FS...

Clever things can now begin to happen. :)

Cautions:

- The holo-shell will run on vty4 so you won't see its output normally
  - Unless you're using a serial console
- The filesystems will be mounted on /mnt/<fsname>
- Hitting ctrl-c in the holo-shell will cause sysinstall to abort!

# What if I don't have PXE?

---

PXE is nice, but many older cards/systems (that make perfectly good diskless clients) don't have it.

Solution: Etherboot

From etherboot web site:

Etherboot is a software package for creating ROM images that can download code over an Ethernet network to be executed on an x86 computer.

Etherboot can be run from a floppy or burned onto an EPROM for use with a supported NIC.

*<http://etherboot.sourceforge.net/>*

There is also a project called 'NILO' that aims to create PXE-compatible boot ROMS.

(Still in its infancy)

*<http://www.nilo.org/>*

# About etherboot

---

Etherboot supports (contains drivers for) lots of network cards:

- 3Com (509,590,90x,980...)
- Intel (EtherExpress, EEPro...)
- DEC (tulip and some clones like davicom)
- VIA-Rhine/RhineII
- Realtek
- Many, many more
- Even some wireless cards!

# Compiling etherboot

---

## Options:

- ▣ `-DASK_BOOT=n`
  - Ask to boot from local disk or network
  - `n` is timeout in seconds (0 for none)
- ▣ `-DANS_DEFAULT=<default>`
  - Set default boot mode for above
    - `ANS_NETWORK` to boot from net, `ANS_LOCAL` to boot from disk
- ▣ `-DNO_DHCP_SUPPORT`
  - Use bootp instead of dhcp
- ▣ `-DREQUIRE_VCI_ETHERBOOT`
  - Require a vendor-encapsulated identifier from DHCP server
  - Adds some safety on a non-dedicated LAN
- ▣ `-DTAGGED_IMAGE`
  - Enable tagged image kernel support (for Linux kernels)
- ▣ `-DELFB_IMAGE`
  - Enable generic ELF kernel support
- ▣ `-DIMAGE_FREEBSD`
  - Enable FreeBSD support (requires `-DELFB_IMAGE`)
- ▣ `-DCONSOLE_SERIAL`
  - Use serial console instead of CRT

# Compiling etherboot (continued)

---

To make a ROM image suitable for burning to EPROM:

- Run `make bin32/<cardtype>.rom`

Note: you may have to use a vendor-provided utility to "turn on" the boot ROM in your ethernet card

To make an image that boots from floppy (for testing):

- Put a floppy in fd0
- Run `make bin32/<cardtype>.fd0`
- Compiles etherboot for your card and writes bootloader to a floppy

Somebody has even put up a web form that automatically returns a ROM image based on form input:

*<http://rom-o-matic.net/>*

# Using etherboot

---

Creating tagged image:

`mknbi-linux` creates a tagged image from a kernel

Ex: `mknbi-linux --output=taggedkernel bzImage`

For FreeBSD, compile etherboot with `-DELFI_IMAGE` and `-DIMAGE_FREEBSD`.

(don't need to make tagged image in this case)

# Making etherboot ignore other DHCP offers

---

The compile option `-DREQUIRE_VCI_ETHERBOOT` makes etherboot ignore DHCP offers that don't include a special VCI.

```
option vendor-encapsulated-options 3c:09:45:74:68:65:72:62:6f:6f:74:ff;
```

This makes ISC-DHCPD send a special VCI that etherboot will recognize.

- This tells etherboot to obey this offer.
- Spells out "Etherboot" :)

# And now for something completely different...

---

Let's install FreeBSD!

# Credit where credit is due

---

Thanks to the following sources for information:

Alfred Perlstein - <http://people.freebsd.org/~alfred/pxe/>

Etherboot-users mailing list - <http://www.geocrawler.com/archives/3/5299/2001/7/100/6129709/>

H. Peter Anvin - <http://syslinux.zytor.com/pxe.php>

Etherboot manual - <http://etherboot.sourceforge.net/doc/html/userman.html>

Rembo Technology (bpbatch) - <http://www.bpbatch.org/>

NILO Project - <http://www.nilo.org/>

Red Hat Software -

<http://www.redhat.com/docs/manuals/linux/RHL-7.3-Manual/custom-guide/ch-kickstart2.html>

The following FreeBSD manpages:

- `sysinstall(8)`
- `loader(8)`
- `pxeboot(8)`
- `dhcpd.conf(5)`

Thanks to Adrian Filipi-Martin for his comments.