

- [Partners](#)
- [Support](#)
- [Community](#)
- [Ubuntu.com](#)

- [Page History](#)
- [Login to edit](#)

DisklessUbuntuHowto

What is diskless booting?

Diskless booting is using a remote system or systems to store the kernel and the filesystem that will be used on other computer(s).

Why do it?

Imagine my case I admin about 25 public workstations for a local library, if they want something changed across the board I can either go sit at each PC and spend X minutes on it, adding up to who knows how many hours, *or* I can simply make the change at any one of the 25 systems and have it affect every system equally. The same goes for updates and many other operations. It really eases support issues.

How is this different than ThinClientHowto?

Thin clients use some of the same principles but they also connect to a remote X session, which means everything runs on the remote server - all applications will consume the servers resources, such as RAM and CPU cycles.

Diskless Booting simply uses the remote server for storage and still runs all applications on the local client station. This works better if you have full powered PC's to work with, and are working with a large number of clients that would require too much CPU and RAM to run all their applications on one server.

It is different enough to have multiple machines mounting the same root filesystem as opposed to simply being a remote monitor and keyboard to warrant a separate how-to I think.

Oliver Grawert says :- *you could have achieved this easier by following the thin client howto, remove the ltsp-client package from the chroot and install ubuntu-desktop (or whatever desktop you want) there, would save you a lot of configuration work ;)*

How does it work?

There are a lot of parallels to [ThinClientHowto](#), diskless booting requires a DHCP server which a bootable PXE network card will query to get its configuration and location of the file to tftp from the server, after booting the PXE image the client will tftp and boot the kernel image(with args specified in the pxe config), those args will tell the kernel how to configure itself, and the path to mount the NFS share where its / directory is located.

Requirements

- An Ubuntu system with (preferably) nfs-kernel-server and tftpd server (the server)
- At least one PXE-bootable system (the client)
- It helps to have the client set up in its final configuration before you start
- Enough disk space on the server to hold the client filesystem
- A fast network connection between the client and the server
- A DHCP server which is capable of supporting PXE clients, or a separate network segment where you can run a dedicated DHCP server
- A good understanding of Linux

Getting Started

Contents

1. [What is diskless booting?](#)
2. [Why do it?](#)
3. [How is this different than ThinClientHowto?](#)
4. [How does it work?](#)
5. [Requirements](#)
6. [Getting Started](#)
 1. [Naming Conventions](#)
 2. [Set up your Server](#)
 3. [Creating your NFS installation](#)
 4. [Alternative to the above method](#)
 5. [Setup your client](#)
7. [Now What?](#)
 1. [Adding a swap file](#)
8. [Static IP](#)
9. [Gotchas](#)
10. [Status](#)
11. [Credits](#)

Naming Conventions

Client: A diskless system that you wish to boot via a network connection

Server: An always-on system which will provide the necessary files to allow the client to boot over the network

Set up your Server

1. Install the required packages

1. Ubuntu 14.04

```
sudo apt-get install isc-dhcp-server tftpd-hpa syslinux nfs-kernel-server initramfs-tools
```

2. Ubuntu 14.10 and after

```
sudo apt-get install isc-dhcp-server tftpd-hpa syslinux pxelinux nfs-kernel-server initramfs-tools
```

2. Configure your DHCP server

You need to set up the DHCP server to offer `/tftpboot/pxelinux.0` as a boot file as a minimum. You also assign a fixed IP to the machine you want to boot with PXE (the client). A range of other options are also available but are beyond the scope of this article. Your `/etc/dhcp/dhcpd.conf` might look like this assuming your subnet is `192.168.2.0`

```
allow booting;
allow bootp;

subnet 192.168.2.0 netmask 255.255.255.0 {
    range 192.168.2.xxx 192.168.2.xxx;
    option broadcast-address 192.168.2.255;
    option routers 192.168.2.xxx;
    option domain-name-servers 192.168.2.xxx;

    filename "/pxelinux.0";
}

# force the client to this ip for pxe.
# This is only necessary assuming you want to send different images to different computers.
host pxe_client {
    hardware ethernet xx:xx:xx:xx:xx:xx;
    fixed-address 192.168.2.xxx;
}
```

NOTE 1: You will need to replace the `'xx:xx:xx:xx:xx:xx'` and the `'192.168.2.xxx'` with your own values

NOTE 2: the filename is a *relative* path to the root of the tftp server.

Restart DHCP Server using the command

```
sudo service isc-dhcp-server restart
```

3. Configure the TFTP Server

We need to set `tftpd-hpa` to run in daemon mode and to use `/tftpboot` as its root directory.

Here is an example `/etc/default/tftpd-hpa` file

```
#Defaults for tftpd-hpa
RUN_DAEMON="yes"
OPTIONS="-l -s /tftpboot"
```

4. Configure your tftp root directory

a. Create directories

```
sudo mkdir -p /tftpboot/pxelinux.cfg
```

b. Copy across bootfile

a. Ubuntu 14.04:

```
sudo cp /usr/lib/syslinux/pxelinux.0 /tftpboot
```

b. Ubuntu 14.10 and after:

```
sudo cp /usr/lib/PXELINUX/pxelinux.0 /tftpboot
sudo mkdir -p /tftpboot/boot
sudo cp -r /usr/lib/syslinux/modules/bios /tftpboot/boot/isolinux
```

c. Create default configuration file `/tftpboot/pxelinux.cfg/default`

```
LABEL linux
KERNEL vmlinuz-2.6.15-23-686
APPEND root=/dev/nfs initrd=initrd.img-2.6.15-23-686 nfsroot=192.168.2.2:/nfsroot ip=dhcp rw
```

NOTE1: your nfs server IP address, kernel name, and initrd name will likely be different. If you have a preconfigured system the names should be the names of the kernel and initrd (see below) on the client system

NOTE2: to find the vmlinuz type `uname -r`

NOTE3: There are more options available such as MAC or IP identification for multiple config files see `syslinux/pxelinux` documentation for help.

NOTE4: Newer distributions might require that you append `,"rw"` to the end of the `"nfsroot="` specification, to prevent a race in the Upstart version of the `statd` and `portmap` scripts.

d. Set permissions

```
sudo chmod -R 777 /tftpboot
```

NOTE:If the files do not have the correct permissions you will receive a "File Not Found" or "Permission Denied" error.

e. Start the `tftp-hpa` service:

```
sudo /etc/init.d/tftpd-hpa start
```

5. Configure OS root

a. Create a directory to hold the OS files for the client

```
sudo mkdir /nfsroot
```

b. configure your `/etc/exports` to export your `/nfsroot`

```
/nfsroot          192.168.2.xxx(rw,no_root_squash,async,insecure)
```

NOTE: The `'192.168.2.xxx'` should be replaced with either the client IP or hostname for single installations, or wildcards to match the set of servers you are using.

Note: In versions prior to Ubuntu 11.04 the option `','insecure'` is not required after `async`.

c. sync your exports

```
sudo exportfs -rv
```

Creating your NFS installation

There are a few ways you can go about this:

- `debbootstrap` (as outlined at [Installation/OnNFSDrive](#))
- copying the install from your server
- install [k]ubuntu on the client from CD, after you've got your system installed and working on the network mount the `/nfsroot` and copy everything from your working system to it.

This tutorial will focus on the last option. The commands in this section should be carried out on the client machine unless it is explicitly noted otherwise. You should ensure that the following package is installed on the client `nfs-common`

1. Copy current kernel version to your home directory.

`uname -r` will print your kernel version, and `~` is shorthand for your home directory.

```
sudo cp /boot/vmlinuz-`uname -r` ~
```

2. Create an `initrd.img` file

a. Change the `BOOT` flag to `nfs` in `/etc/initramfs-tools/initramfs.conf`

```
#
# BOOT: [ local | nfs ]
#
# local - Boot off of local media (harddrive, USB stick).
#
# nfs - Boot using an NFS drive as the root of the drive.
#
```

```
BOOT=nfs
```

b. Change the `MODULES` flag to `netboot` in `/etc/initramfs-tools/initramfs.conf`

```
#
# MODULES: [ most | netboot | dep | list ]
#
# most - Add all framebuffer, acpi, filesystem, and harddrive drivers.
#
# dep - Try and guess which modules to load.
#
# netboot - Add the base modules, network modules, but skip block devices.
#
# list - Only include modules from the 'additional modules' list
#
```

```
MODULES=netboot
```

NOTE: if you have anything in `/etc/initramfs-tools/conf.d/driver-policy`, this line will be ignored.

- c. Check which modules you will need for your network adapters and put their names into `/etc/initramfs-tools/modules` (for example `forcedeth`, `r8169` or `tulip`)
- d. Run `mkinitramfs`

```
mkinitramfs -o ~/initrd.img-`uname -r`
```

3. Copy OS files to the server

```
mount -t nfs -onolock 192.168.1.2:/nfsroot /mnt
cp -ax /. /mnt/
cp -ax /dev/. /mnt/dev/
```

NOTE: If the client source installation you copied the files from should remain bootable and usable from local hard disk, restore the former `BOOT=local` and `MODULES=most` options you changed in `/etc/initramfs-tools/initramfs.conf`. Otherwise, the first time you update the kernel image on the originating installation, the `initram` will be built for network boot, giving you "can't open `/tmp/net-eth0.conf`" and "kernel panic". Skip this step if you no longer need the source client installation.

4. Copy kernel and `initrd` to tftp root.

Run these commands **ON THE SERVER**

```
sudo cp ~/vmlinuz-`uname -r` /tftpboot/
sudo cp ~/initrd.img-`uname -r` /tftpboot/
```

5. Modify `/nfsroot/etc/network/interfaces`

When booting over the network, the client will already have carried out a DHCP discovery before the OS is reached. For this reason you should ensure the OS does not try to reconfigure the interface later.

You should set your network interface to be "manual" not "auto" or "dhcp". Below is an example file.

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).
```

```
# The loopback network interface
auto lo
iface lo inet loopback
```

```
# The primary network interface, commented out for NFS root
#auto eth0
#iface eth0 inet dhcp
iface eth0 inet manual
```

NOTE: For Ubuntu 7.04 (Feisty Fawn) it seems the `/etc/network/interfaces` needs a little tweak, in order *not* to have the [NetworkManager](#) fiddle with the interface since it's already configured (see also bug #111227 : "NFS-root support indirectly broken in Feisty")

NOTE: If you use `dnsmasq` for your boot client (instead of `dhcp`), you will need to provide `nameserver(s)` information in `/etc/resolvconf/resolv.conf.d/base`. For instance, `nameserver 192.168.0.xxx` (You will need to replace the '192.168.0.xxx' with your own values.)

6. Configure `fstab`

`/nfsroot/etc/fstab` contains the information the client will use to mount file systems on boot, edit it to ensure it looks something like this ('note no swap')

```
# /etc/fstab: static file system information.
#
# <file system> <mount point> <type> <options> <dump> <pass>
proc /proc proc defaults 0 0
/dev/nfs / nfs defaults 1 1
none /tmp tmpfs defaults 0 0
none /var/run tmpfs defaults 0 0
none /var/lock tmpfs defaults 0 0
none /var/tmp tmpfs defaults 0 0
/dev/hdc /media/cdrom0 udf,iso9660 user,noauto 0 0
```

NOTE: if you have entries for other `tmpfs` that's fine to leave them in there


7. Disable Grub update


Since diskless systems don't need grub to boot, disable update scripts to prevent glitches during further software updates. Comment out `exec update-grub` in `/etc/kernel/postinst.d/zz-update-grub`

Alternative to the above method

An alternative to parts 1, 2, 3 and 4 exists, if you are copying your `nfsroot` to the same server as your tftp server. To use the alternative,

do not copy your `vmlinuz-2.##-generic` file to your home directory. Do part 2(a) and then do `sudo mkinitramfs -o /boot/initrd.img-X` where of course X is your kernel version ('2.6.20-16-generic' from the `mkinitramfs` example shown above). Do part 3.

On The Server Instead of copying your kernel and `initrd` into the `tftp` folder as suggested in part 4, `cd` into the `boot` folder in the root of the machine that you just copied over in part 3. If you followed part 3 exactly, this will be in `/nfsroot/boot/`. Create a symlink called '`vmlinuz`' what points to your current kernel version in that same directory. Create another symlink called '`initrd.img`' that points to the `initrd.img` that you are currently using (same version as your kernel). Create one more symlink called '`config`' that points to the `config` you are using. You will likely need to create these symlinks with `sudo`. They will help keep your `tftp` configuration simple and can help ease system upgrades, all you have to do is upgrade the '`vmlinuz`', '`initrd.img`', and '`config`' symlinks to point to the new files. `cd` into your `tftp` directory, and create symlinks  to the '`vmlinuz`', '`initrd.img`', and '`config`' symlinks you just created. If you will be doing this for more than one machine, you may want to add the name of the machine to the symlink in this directory so that you can tell them apart. From here, continue at part 5 of the above instructions. When you finish the from part 5 onward of the above instructions you should have a booting machine in the same state as you would had you followed them from part 1 onwards.


 - Remember that it will not work if creating symlink out of `chroot`. Symlink won't work if you keep the default `-s` (`--secure`) option from `tftpd`, which actually does `chroot`. You have to do something else to make those files (`vmlinuz`, `initrd.img`, `config`) visible, for example when you use different filesystem for storing `nfsroot`: `mount -o bind /mnt/nfsroot/boot /tftpboot/ubuntuboot`, then add relative path '`./ubuntuboot`' to `KERNEL` file definition

Setup your client

Enter your BIOS settings and configure your system to boot from LAN

If you have options for different LAN boot methods choose PXE

Now What?

Reboot your client PC it should get its net config from the DHCP server, download the `pxelinux` image, download and boot the kernel, mount your root fs and continue to boot just as if it were using its internal HD, only now you can yank that HD out and use it for something else, that machine doesn't need it any more 

Adding a swap file

In case you do need or want to set up a swapfile, here's what I did to get one working:

```
sudo apt-get install dphys-swapfile
```

this package sets up a swap file at `/var/swap` that is 2x your current ram. however, it still doesn't setup the swapfile on its own, though it does try, to get the swap file working the rest of the way, do:

```
sudo losetup /dev/loop0 /var/swap
sudo swapon /dev/loop0
```

then, run `top` and you will see you have a swap file. however, put in as much ram as you need for what you are going to run, and just look at your swap file as "in case of emergency", because it is not efficient as a ram or a regular swap file, but should keep something from crashing. I'm running `mythtv` on top of a full feisty desktop, and tried it with 256mb ram, and `myth-frontent` would crash when i would try to bring it up. with the swap file, it would launch, but would take a while. i added an additional 256mb to bring the total to 512mb and it runs flawless, and i add the swap file just in case. i just make a script for it to set up the swap file at boot.

Static IP

As described in #175324 `ipconfig` have problems with DHCP relays. Static IP is configured by parameter:

```
ip=<client-ip>:<server-ip>:<gw-ip>:<netmask>:<hostname>:<device>:<autoconf>
```

<client-ip> IP address of the client. If empty, the address will

either be determined by RARP/BOOTP/DHCP. What protocol

is used de- pends on the <autoconf> parameter. If this parameter is not empty, `autoconf` will be used.

<server-ip> IP address of the NFS server. If RARP is used to

determine the client address and this parameter is NOT empty only replies from the specified server are accepted. To use different RARP and NFS server, specify your RARP server here (or leave it blank), and specify your NFS server in the '`nfsroot`' parameter (see above). If this entry is blank the address of the server is used which answered the RARP/BOOTP/DHCP request.

<gw-ip> IP address of a gateway if the server is on a different

subnet. If this entry is empty no gateway is used and the server is assumed to be on the local network, unless a value has

been received by BOOTP/DHCP.

<netmask> Netmask for local network interface. If this is empty,

the netmask is derived from the client IP address assuming classful addressing, unless overridden in BOOTP/DHCP reply.

<hostname> Name of the client. If empty, the client IP address is

used in ASCII notation, or the value received by BOOTP/DHCP.

<device> Name of network device to use. If this is empty, all


devices are used for RARP/BOOTP/DHCP requests, and the first one we receive a reply on is configured. If you have only one device, you can safely leave this blank.

<autoconf> Method to use for autoconfiguration. If this is either

'rarp', 'bootp', or 'dhcp' the specified protocol is used. If the value is 'both', 'all' or empty, all protocols are used. 'off', 'static' or 'none' means no autoconfiguration.

Gotchas

- This does not work for a PXE server running 10.04.1 LTS, nor does it work for clients trying to run 10.04.1 LTS. The client receives an offer from DHCP, gets the kernel, and fails while trying to load.

 **NOTE:** It WILL work if you add required module names in `/etc/initramfs-tools/modules` (module names for your network adapters, like `forcedeth` or `tulip`)

- Remember to use `-o nolock` when first mounting your NFS or have portmapper running or you'll end up with a long (nearly 2 min) hang mounting the remote NFS. You may also be able to get around this by mounting your `nfsroot` in read/write mode off the bat, by adding `,"rw"` to the `"nfsroot="` option on the kernel command line.
- If you are going to have multiple systems share this root you need to make sure some directories like `/var/run`, `/var/lock` and `/tmp` are mounted in `tmpfs` so different systems won't be conflicting into each other. **NOTE:** it is not required on Ubuntu 15.04+ though, `/var/run` and `/var/lock` are already mounted using `tmpfs`.
 - You probably also want to mount `/media` in `tmpfs` space if you don't want every workstation having access to every other workstations storage media.
- Think of things in cron like `slocate` - do you really want 25 computers trying to `updatedb` at 4AM all on the same HD?
- Don't forget graphics drivers, conflicts can arise if your machines use varying graphics cards.
- Do you really want all of the machines trying to append to the same log file? Maybe have each machine send their logs to a central log server, or mount the log directories over NFS.
- With this setup under 8.10 to get any NFS directories to mount in the `fstab` on boot, you need to add `ASYNCMOUNTNFS=no` to `/etc/default/rcS` (because otherwise `/etc/init.d/mountnfs.sh` expects `/etc/network/ifup.d/` scripts to take care of it (which don't run for manual interfaces))
- If you intend to use AutoFS to automatically mount/unmount directories on a diskless system, you may run into this [small bump in the road](#). There is a one-liner fix in the `autofs` `init.d` script to get you going again.
- If you are performing upgrades on your diskless client and run into [these problems](#), you can fix them by commenting out one line in a config file, as outlined on the same page.
- Beware of poorly programed applications which crash hard with errors indicating there is already another instance of itself running if you ever have a hard crash and have to use the `reboot/power` button. Many applications, including Firefox, Thunderbird, Skype and many others don't handle such crashes cleanly and will give you such errors if you ever find yourself in this situation. To remedy it, you must `_manually_` go through their config folders (for example, firefox uses `~/.mozilla/firefox/XXX.default/.`) and delete all of the locks. For Firefox and related browsers, you may also have to delete several of the `.sqlite` files in order to get your history, in-browser search bar and such features to work properly again; as for which ones, you'll have to use trial and error to figure it out (or, `rm -rf *.sqlite` does it too, but assures you obliterate everything at the same time). Also while your looking for locks to remove in the respective config folders, look for and delete files such as `.nfs00000000e000e47e0000000a` ; These were left by the crashed instance of the application and do not directly get in your way but its cruft you don't want sitting around.

Status

This [HowTo](#) is confirmed to work with Ubuntu 12.04.

This [HowTo](#) is confirmed to work with Lubuntu 12.04 on Intel Atom client systems.

This [HowTo](#) is confirmed to work with Ubuntu 14.04.2 LTS when the following line is added to `/tftpboot/pxelinux.cfg/default`:

```
DEFAULT linux
```

This [HowTo](#) is confirmed to work with Ubuntu 15.04

Credits

For the feisty interfaces fix and locating your current kernel: http://developer.novell.com/wiki/index.php/HOWTO:_Convert_Ubuntu_to_Diskless For the tftp-hpa config - dodox in the forums: <http://ubuntuforums.org/showthread.php?t=236518>

CategoryLtsp

DisklessUbuntuHowto (last edited 2015-10-26 19:49:48 by [doug-manley](#) @ 38.98.198.34[38.98.198.34]:doug-manley)