Remote access

Introduction to remote access

Edit this on GitHub

Sometimes you need to access a Raspberry Pi without connecting it to a monitor. Perhaps the Raspberry Pi is embedded in something like a robot, or you want to view some information from it from elsewhere. Or perhaps you simply don't have a spare monitor!

You can connect to your Raspberry Pi from another machine. But in order to do so you'll need to know its IP address.

Any device connected to a Local Area Network is assigned an IP address. In order to connect to your Raspberry Pi from another machine using SSH or VNC, you need to know the Raspberry Pi's IP address. This is easy if you have a display connected, and there are a number of methods for finding it remotely from another machine on the network.

How to find your IP address

Edit this on GitHub

To find the local IP address of your Raspberry Pi, use one of the following methods.

Desktop

Hover over the network icon in the system tray, and a tooltip will appear. This tooltip displays the name of the network you're currently connected to and your IP address.

Command line

Run the following command to output your IP address to the command line:

```
hostname -I
```

Boot output

If you use a display with your Raspberry Pi and you boot to the command line instead of the desktop, the boot sequence will include your IP address as one of the last few output messages before your login prompt.

Network Manager

You can use the built-in Network Manager CLI (`nmcli`) to access details about your network. Run the following command:

```
nmcli device show
```

You should see output similar to the following:

```
GENERAL.DEVICE:                         wlan0
GENERAL.TYPE:                           wifi
GENERAL.HWADDR:                         D0:3B:FF:41:AB:8A
GENERAL.MTU:                            1500
GENERAL.STATE:                          100 (connected)
GENERAL.CONNECTION:                     exampleNetworkName
```

```
GENERAL.CON-PATH:                       /org/freedesktop/NetworkManager/ActiveC
onnection/2
IP4.ADDRESS[1]:                         192.168.1.42/24
IP4.GATEWAY:                            192.168.1.1
IP4.ROUTE[1]:                           dst = 192.168.1.0/24, nh = 0.0.0.0, mt
= 600
IP4.ROUTE[2]:                           dst = 0.0.0.0/0, nh = 192.168.1.1, mt =
600
IP4.DNS[1]:                             192.168.1.3
IP6.ADDRESS[1]:                         ab80::11ab:b1fc:bb7e:a8a5/64
IP6.GATEWAY:                            --
IP6.ROUTE[1]:                           dst = ab80::/64, nh = ::, mt = 1024

GENERAL.DEVICE:                         lo
GENERAL.TYPE:                           loopback
GENERAL.HWADDR:                         00:00:00:00:00:00
GENERAL.MTU:                            65536
GENERAL.STATE:                          100 (connected (externally))
GENERAL.CONNECTION:                     lo
GENERAL.CON-PATH:                       /org/freedesktop/NetworkManager/ActiveC
onnection/1
IP4.ADDRESS[1]:                         127.0.0.1/8
IP4.GATEWAY:                            --
IP6.ADDRESS[1]:                         ::1/128
IP6.GATEWAY:                            --

GENERAL.DEVICE:                         p2p-dev-wlan0
GENERAL.TYPE:                           wifi-p2p
GENERAL.HWADDR:                         (unknown)
GENERAL.MTU:                            0
GENERAL.STATE:                          30 (disconnected)
GENERAL.CONNECTION:                     --
GENERAL.CON-PATH:                       --

GENERAL.DEVICE:                         eth0
GENERAL.TYPE:                           ethernet
GENERAL.HWADDR:                         D0:3B:FF:41:AB:89
GENERAL.MTU:                            1500
GENERAL.STATE:                          20 (unavailable)
GENERAL.CONNECTION:                     --
GENERAL.CON-PATH:                       --
WIRED-PROPERTIES.CARRIER:               off
IP4.GATEWAY:                            --
IP6.GATEWAY:                            --
```

This command outputs information about the various network interfaces accessible on your Raspberry Pi. Check the `GENERAL.TYPE` row to see which kind of network interface each block describes. For example, "ethernet" is the Ethernet port on your device, and "wifi" refers to the Wi-Fi chip built into some devices. You'll look at different blocks of output to find your IP address depending on the way your device accesses the internet:

- if your device connects to the internet using Wi-Fi, check the "wifi" block
- if your device connects to the internet using the Ethernet port, check the "ethernet" block

Once you've identified the correct network interface block, look for a field named `IP4.ADDRESS[1]` for an IPv4 address or `IP6.ADDRESS[1]` for an IPv6 address. You can ignore the trailing slash and number (e.g. `/24`) in those fields.

In the example above, the Raspberry Pi uses Wi-Fi to access the internet. Check the block where the `GENERAL.TYPE` field reads "wifi" to find the IP address. In this case, you can access this device using the IPv4 address in the `IP4.ADDRESS[1]` field: `192.168.1.42`.

Resolving `raspberrypi.local` with mDNS

On Raspberry Pi OS, multicast DNS is supported out-of-the-box by the Avahi service.

If your device supports mDNS, you can reach your Raspberry Pi by using its hostname and the `.local` suffix. The default hostname on a fresh Raspberry Pi OS install is `raspberrypi`, so by default any Raspberry Pi running Raspberry Pi OS responds to:

```
ping raspberrypi.local
```

If the Raspberry Pi is reachable, `ping` will show its IP address:

```
PING raspberrypi.local (192.168.1.131): 56 data bytes
64 bytes from 192.168.1.131: icmp_seq=0 ttl=255 time=2.618 ms
```

If you change the system hostname of the Raspberry Pi (e.g., by editing `/etc/hostname`), Avahi also changes the `.local` mDNS address.

If you don't remember the hostname of the Raspberry Pi, but have a system with Avahi installed, you can browse all the hosts and services on the LAN with the `avahi-browse` command.

Router devices list

In a web browser, navigate to your router's IP address. This is often `http://192.168.1.1`, but you may be able to find it printed on a label on your router. This will take you to a control panel. Then log in using your credentials, which is usually also printed on the router or sent to you in the accompanying paperwork. Browse to the list of connected devices or similar (all routers are different), and you should see some devices you recognise. Some devices are detected as PCs, tablets, phones, printers, etc. so you should recognise some and rule them out to figure out which is your Raspberry Pi. Also note the connection type; if your Raspberry Pi is connected with a wire there should be fewer devices to choose from.

nmap command

The `nmap` command (Network Mapper) is a free and open-source tool for network discovery, available for Linux, macOS, and Windows.

- To install on **Linux**, install the `nmap` package e.g. `apt install nmap`.
- To install on **macOS** or **Windows**, see the nmap.org download page.

To use `nmap` to scan the devices on your network, you need to know the subnet you are connected to. First find your own IP address, in other words the one of the computer you're using to find your Raspberry Pi's IP address:

- On **Linux**, type `hostname -I` into a terminal window
- On **macOS**, go to `System Preferences` then `Network` and select your active network connection to view the IP address
- On **Windows**, go to the Control Panel, then under `Network and Sharing Center`, click `View network connections`, select your active network connection and click `View status of this connection` to view the IP address

Now you have the IP address of your computer, you will scan the whole subnet for other devices. For example, if your IP address is `192.168.1.5`, other devices will use addresses like `192.168.1.2`, `192.168.1.6`, `192.168.1.20`, etc. The notation of this subnet range is `192.168.1.0/24` (this covers `192.168.1.0` to `192.168.1.255`).

Now use the `nmap` command with the `-sn` flag (ping scan) on the whole subnet range. This may take a few seconds:

```
nmap -sn 192.168.1.0/24
```

Ping scan just pings all the IP addresses to see if they respond. For each device that responds to the ping, the output shows the hostname and IP address like so:

```
Starting Nmap 6.40 ( http://nmap.org ) at 2014-03-10 12:46 GMT
Nmap scan report for hpprinter (192.168.1.2)
Host is up (0.00044s latency).
Nmap scan report for Gordons-MBP (192.168.1.4)
Host is up (0.0010s latency).
Nmap scan report for ubuntu (192.168.1.5)
Host is up (0.0010s latency).
Nmap scan report for raspberrypi (192.168.1.8)
Host is up (0.0030s latency).
Nmap done: 256 IP addresses (4 hosts up) scanned in 2.41 seconds
```

Here you can see a device with hostname `raspberrypi` has IP address `192.168.1.8`. Note, to see the hostnames, you must run nmap as root by prepending `sudo` to the command.

Getting the IP address of a Raspberry Pi using your smartphone

The Fing app is a free network scanner for smartphones. It is available for Android and iOS.

Your phone and your Raspberry Pi have to be on the same network, so connect your phone to the correct wireless network.

When you open the Fing app, touch the refresh button in the upper right-hand corner of the screen. After a few seconds you will get a list with all the devices connected to your network. Scroll down to the entry with the manufacturer "Raspberry Pi". You will see the IP address in the bottom left-hand corner, and the MAC address in the bottom right-hand corner of the entry.

Setting up an SSH Server

Edit this on GitHub

You can access the command line of a Raspberry Pi remotely from another computer or device on the same network using the Secure Shell (SSH) protocol.

You will only have access to the command line, not the full desktop environment. For a full remote desktop, see VNC.

Set up your Local Network

Make sure your Raspberry Pi is properly set up and connected. If you are using wireless networking, this can be enabled via the desktop user interface, or using from the command line. If you are not using wireless connectivity, plug your Raspberry Pi directly into the router.

NOTE

> You will need to note down the IP address of your Raspberry Pi in order to connect to it later. Using the `ifconfig` command will display information about the current network status, including the IP address, or you can use `hostname -I` to display the IP addresses associated with the device.

Enabling the Server

Raspberry Pi OS has the SSH server disabled by default. It can be enabled manually from the desktop:

1. Launch `Raspberry Pi Configuration` from the `Preferences` menu
2. Navigate to the `Interfaces` tab
3. Select `Enabled` next to `SSH`
4. Click `OK`

Alternatively you can enable it from the terminal using the raspi-config application,

1. Enter `sudo raspi-config` in a terminal window

2. Select `Interfacing Options`
3. Navigate to and select `SSH`
4. Choose `Yes`
5. Select `Ok`
6. Choose `Finish`

NOTE

For headless setup, SSH can be enabled by placing a file named `ssh`, without any extension, onto the boot partition of the SD Card. When the Raspberry Pi boots, it looks for the `ssh` file. If it is found, SSH is enabled and the file is deleted. The content of the file does not matter; it could contain text, or nothing at all.

NOTE

For headless setup in addition to the `ssh` file you need a `userconf.txt` file, which contains a string `username:encryptedpassword`. Please refer to the section on configuring a user in the discussions around headless setup of a Raspberry Pi.

WARNING

When enabling SSH on a Raspberry Pi that may be connected to the internet, you should ensure that your password is not easily brute forced.

Secure Shell from Linux or Mac OS

Edit this on GitHub

You can use SSH to connect to your Raspberry Pi from a Linux desktop, another Raspberry Pi, or from an Apple Mac without installing additional software.

Open a terminal window on your computer replacing `<IP>` with the IP address of the Raspberry Pi you're trying to connect to and replacing the `<username>` placeholder with the username of your primary user account:

```
ssh <username>@<IP>
```

When the connection works you will see a security/authenticity warning. Type `yes` to continue. You will only see this warning the first time you connect.

NOTE

If you receive a `connection timed out` error it is likely that you have entered the wrong IP address for the Raspberry Pi.

WARNING

> In the event your Raspberry Pi has taken the IP address of a device to which your computer has connected before (even if this was on another network), you may be given a warning and asked to clear the record from your list of known devices. Following this instruction and trying the `ssh` command again should be successful.

Next, enter your account password when prompted.

You should now be able to see the Raspberry Pi prompt, which will be identical to the one found on the Raspberry Pi itself.

```
<username>@raspberrypi ~ $
```

You are now connected to the Raspberry Pi remotely, and can execute commands.

Forwarding X11

You can also forward your X session over SSH, to allow the use of graphical applications, by using the `-Y` flag:

```
ssh -Y <username>@192.168.1.5
```

NOTE

> X11 is no longer installed by default on macOS, so you will have to download and install it.

Now you are on the command line as before, but you have the ability to open up graphical windows. For example, typing:

```
geany &
```

will open up the Geany editor in a window on your local desktop.

Secure Shell from Windows 10

Edit this on GitHub

You can use SSH to connect to your Raspberry Pi from a Windows 10 computer that is using *October 2018 Update* or later without having to use third-party clients.

Open a terminal window on your computer replacing `<IP>` with the IP address of the Raspberry Pi you're trying to connect to,

```
ssh pi@<IP>
```

When the connection works you will see a security/authenticity warning. Type `yes` to continue. You will only see this warning the first time you connect.

NOTE

If you receive a `connection timed out` error it is likely that you have entered the wrong IP address for the Raspberry Pi.

WARNING

In the event your Raspberry Pi has taken the IP address of a device to which your computer has connected before (even if this was on another network), you may be given a warning and asked to clear the record from your list of known devices. Following this instruction and trying the `ssh` command again should be successful.

Next you will be prompted for the password for the `pi` login: the default password on Raspberry Pi OS is `raspberry`.

For security reasons it is highly recommended to change the default password on the Raspberry Pi (also, you can not login through ssh if the password is blank). You should now be able to see the Raspberry Pi prompt, which will be identical to the one found on the Raspberry Pi itself.

If you have set up another user on the Raspberry Pi, you can connect to it in the same way, replacing the username with your own, e.g. `eben@192.168.1.5`

```
pi@raspberrypi ~ $
```

You are now connected to the Raspberry Pi remotely, and can execute commands.

Passwordless SSH Access

Edit this on GitHub

It is possible to configure your Raspberry Pi to allow access from another computer without needing to provide a password each time you connect. To do this, you need to use an SSH key instead of a password. To generate an SSH key:

Checking for Existing SSH Keys

First, check whether there are already keys on the computer you are using to connect to the Raspberry Pi:

```
ls ~/.ssh
```

If you see files named `id_rsa.pub` or `id_dsa.pub` then you have keys set up already, so you can skip the 'Generate new SSH keys' step below.

Generate new SSH Keys

To generate new SSH keys enter the following command:

```
ssh-keygen
```

Upon entering this command, you will be asked where to save the key. We suggest saving it in the default location (`~/.ssh/id_rsa`) by pressing `Enter`.

You will also be asked to enter a passphrase, which is optional. The passphrase is used to encrypt the private SSH key, so that if someone else copied the key, they could not impersonate you to gain access. If you choose to use a passphrase, type it here and press `Enter`, then type it again when prompted. Leave the field empty for no passphrase.

Now look inside your `.ssh` directory:

```
ls ~/.ssh
```

and you should see the files `id_rsa` and `id_rsa.pub`:

```
authorized_keys  id_rsa  id_rsa.pub  known_hosts
```

The `id_rsa` file is your private key. Keep this on your computer.

The `id_rsa.pub` file is your public key. This is what you share with machines that you connect to: in this case your Raspberry Pi. When the machine you try to connect to matches up your public and private key, it will allow you to connect.

Take a look at your public key to see what it looks like:

```
cat ~/.ssh/id_rsa.pub
```

It should be in the form:

```
ssh-rsa <REALLY LONG STRING OF RANDOM CHARACTERS> user@host
```

Copy your Key to your Raspberry Pi

Using the computer which you will be connecting from, append the public key to your `authorized_keys` file on the Raspberry Pi by sending it over SSH:

```
ssh-copy-id <USERNAME>@<IP-ADDRESS>
```

NOTE

During this step you will need to authenticate with your password.

Alternatively, if `ssh-copy-id` is not available on your system, you can copy the file manually over SSH:

```
cat ~/.ssh/id_rsa.pub | ssh <USERNAME>@<IP-ADDRESS> 'mkdir -p ~/.ssh && cat >>
~/.ssh/authorized_keys'
```

If you see the message `ssh: connect to host <IP-ADDRESS> port 22: Connection refused` and you know the `IP-ADDRESS` is correct, then you may not have enabled SSH on your Raspberry Pi. Run `sudo raspi-config` in the Raspberry Pi's terminal window, enable SSH, then try to copy the files again.

Now try `ssh <USER>@<IP-ADDRESS>` and you should connect without a password prompt.

If you see a message "Agent admitted failure to sign using the key" then add your RSA or DSA identities to the authentication agent `ssh-agent` then execute the following command:

```
ssh-add
```

NOTE

You can also send files over SSH using the `scp` (secure copy) command.

Adjust Directory Permissions

If you can't establish a connection after following the steps above there might be a problem with your directory permissions. First, you want to check the logs for any errors:

```
journalctl -f
# might return:
Nov 23 12:31:26 raspberrypi sshd[9146]: Authentication refused: bad ownership o
r modes for directory /home/pi
```

If the log says `Authentication refused: bad ownership or modes for directory /home/pi` there is a permission problem regarding your home directory. SSH needs your home and `~/.ssh` directory to not have group write access. You can adjust the permissions using `chmod`:

```
chmod g-w $HOME
chmod 700 $HOME/.ssh
chmod 600 $HOME/.ssh/authorized_keys
```

Now only the user itself has access to `.ssh` and `.ssh/authorized_keys` in which the public keys of your remote machines are stored.

Storing the passphrase in the macOS keychain

If you are using macOS, and after verifying that your new key allows you to connect, you have the option of storing the passphrase for your key in the macOS keychain. This allows you to connect to your Raspberry Pi without entering the passphrase.

Run the following command to store it in your keychain:

```
ssh-add -K ~/.ssh/id_rsa
```

NOTE

> From macOS Monterey onwards the `-K` flag has been deprecated and been replaced by the `--apple-use-keychain` flag.

Using Secure Copy

Edit this on GitHub

Secure Copy (`scp`) is a command for sending files over SSH. This means you can copy files between computers, say from your Raspberry Pi to your desktop or laptop, or vice-versa.

First of all, you'll need to know your Raspberry Pi's IP address.

Copying Files to your Raspberry Pi

Copy the file `myfile.txt` from your computer to the `pi` user's home folder of your Raspberry Pi at the IP address `192.168.1.3` with the following command:

```
scp myfile.txt pi@192.168.1.3:
```

Copy the file to the `/home/pi/project/` directory on your Raspberry Pi (the `project` folder must already exist):

```
scp myfile.txt pi@192.168.1.3:project/
```

Copying Files from your Raspberry Pi

Copy the file `myfile.txt` from your Raspberry Pi to the current directory on your other computer:

```
scp pi@192.168.1.3:myfile.txt .
```

Copying Multiple Files

Copy multiple files by separating them with spaces:

```
scp myfile.txt myfile2.txt pi@192.168.1.3:
```

Alternatively, use a wildcard to copy all files matching a particular search with:

```
scp *.txt pi@192.168.1.3:
```

(all files ending in `.txt`)

```
scp m* pi@192.168.1.3:
```

(all files starting with `m`)

```
scp m*.txt pi@192.168.1.3:
```

(all files starting with `m` and ending in `.txt`)

NOTE

Some of the examples above will not work for file names containing spaces. Names like this need to be enclosed in quotes:

```
scp "my file.txt" pi@192.168.1.3:
```

Copying a Whole Directory

Copy the directory `project/` from your computer to the `pi` user's home folder of your Raspberry Pi at the IP address `192.168.1.3` with the following command:

```
scp -r project/ pi@192.168.1.3:
```

Using `rsync`

Edit this on GitHub

You can use the tool `rsync` to synchronise folders between computers. You might want to transfer some files from your desktop computer or laptop to your Raspberry Pi, for example, and for them to be kept up to date, or you might want the pictures taken by your Raspberry Pi transferred to your computer automatically.

Using `rsync` over SSH allows you to transfer files to your computer automatically.

Here is an example of how to set up the sync of a folder of pictures on your Raspberry Pi to your computer:

On your computer, create a folder called `camera`:

```
mkdir camera
```

Look up the Raspberry Pi's IP address by logging in to it and running `hostname -I`. In this example, the Raspberry Pi is creating a timelapse by capturing a photo every minute, and saving the picture with a timestamp in the local folder `camera` on its SD card.

Now run the following command (substituting your own Raspberry Pi's IP address):

```
rsync -avz -e ssh pi@192.168.1.10:camera/ camera/
```

This will copy all files from the Raspberry Pi's `camera` folder to your computer's new `camera` folder.

Network File System (NFS)

Edit this on GitHub

Network File System (NFS) allows you to share a directory located on one networked computer with other computers or devices on the same network. The computer where the directory is located is called the **server**, and computers or devices connecting to that server are called clients. Clients usually `mount` the shared directory to make it a part of their

own directory structure. The shared directory is an example of a shared resource or network share.
For smaller networks, an NFS is perfect for creating a simple NAS (Network-attached storage) in a Linux/Unix environment.

An NFS is perhaps best suited to more permanent network-mounted directories, such as `/home` directories or regularly-accessed shared resources. If you want a network share that guest users can easily connect to, Samba is better suited to the task. This is because tools to temporarily mount and detach from Samba shares are more readily available across old and proprietary operating systems.

Before deploying an NFS, you should be familiar with:

- Linux file and directory permissions
- mounting and unmounting filesystems

Setting up a Basic NFS Server

Install the packages required using the command below:

```
sudo apt install nfs-kernel-server
```

For easier maintenance, we will isolate all NFS exports in single directory, into which the real directories will be mounted with the `--bind` option.

Suppose we want to export our users' home directories, which are in `/home/users`. First we create the export filesystem:

```
sudo mkdir -p /export/users
```

Note that `/export` and `/export/users` will need 777 permissions, as we will be accessing the NFS share from the client without LDAP/NIS authentication. This will not apply if using authentication (see below). Now mount the real `users` directory with:

```
sudo mount --bind /home/users /export/users
```

To save us from retyping this after every reboot, we add the following line to `/etc/fstab`:

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

There are three configuration files that relate to an NFS server:

1. `/etc/default/nfs-kernel-server`

2. `/etc/default/nfs-common`
3. `/etc/exports`

The only important option in `/etc/default/nfs-kernel-server` for now is `NEED_SVCGSSD`. It is set to `"no"` by default, which is fine, because we are not activating NFSv4 security this time.

In order for the ID names to be automatically mapped, the file `/etc/idmapd.conf` must exist on both the client and the server with the same contents and with the correct domain names. Furthermore, this file should have the following lines in the `Mapping` section:

```
[Mapping]

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

However, note that the client may have different requirements for the Nobody-User and Nobody-Group. For example, on RedHat variants, it is `nfsnobody` for both. If you're not sure, check via the following commands to see if `nobody` and `nogroup` are there:

```
cat /etc/passwd
cat /etc/group
```

This way, server and client do not need the users to share same UID/GUID. For those who use LDAP-based authentication, add the following lines to the `idmapd.conf` of your clients:

```
[Translation]

Method = nsswitch
```

This will cause `idmapd` to know to look at `nsswitch.conf` to determine where it should look for credential information. If you have LDAP authentication already working, `nsswitch` shouldn't require further explanation.

To export our directories to a local network `192.168.1.0/24`, we add the following two lines to `/etc/exports`:

```
/export       192.168.1.0/24(rw,fsid=0,insecure,no_subtree_check,async)
/export/users 192.168.1.0/24(rw,nohide,insecure,no_subtree_check,async)
```

Portmap lockdown (optional)

The files on your NFS are open to anyone on the network. As a security measure, you can restrict access to specified clients.

Add the following line to `/etc/hosts.deny`:

```
rpcbind mountd nfsd statd lockd rquotad : ALL
```

By blocking all clients first, only clients in `/etc/hosts.allow` (added below) will be allowed to access the server.

Now add the following line to `/etc/hosts.allow`:

```
rpcbind mountd nfsd statd lockd rquotad : <list of IPv4s>
```

where `<list of IPv4s>` is a list of the IP addresses of the server and all clients. (These have to be IP addresses because of a limitation in `rpcbind`, which doesn't like hostnames.) Note that if you have NIS set up, you can just add these to the same line.

Please ensure that the list of authorised IP addresses includes the `localhost` address (`127.0.0.1`), as the startup scripts in recent versions of Ubuntu use the `rpcinfo` command to discover NFSv3 support, and this will be disabled if `localhost` is unable to connect.

Finally, to make your changes take effect, restart the service:

```
sudo systemctl restart nfs-kernel-server
```

Configuring an NFS Client

Now that your server is running, you need to set up any clients to be able to access it. To start, install the required packages:

```
sudo apt install nfs-common
```

On the client, we can mount the complete export tree with one command:

```
mount -t nfs -o proto=tcp,port=2049 <nfs-server-IP>:/ /mnt
```

You can also specify the NFS server hostname instead of its IP address, but in this case you need to ensure that the hostname can be resolved to an IP on the client side. A robust way of ensuring that this will always resolve is to use the `/etc/hosts` file.

Note that `<nfs-server-IP>:/export` is not necessary in NFSv4, as it was in NFSv3. The root export `:/` defaults to export with `fsid=0`.

We can also mount an exported subtree with:

```
mount -t nfs -o proto=tcp,port=2049 <nfs-server-IP>:/users /home/users
```

To ensure this is mounted on every reboot, add the following line to `/etc/fstab`:

```
<nfs-server-IP>:/   /mnt   nfs    auto  0  0
```

If, after mounting, the entry in `/proc/mounts appears` as `<nfs-server-IP>://` (with two slashes), then you might need to specify two slashes in `/etc/fstab`, or else `umount` might complain that it cannot find the mount.

Portmap lockdown (optional)

Add the following line to `/etc/hosts.deny`:

```
rpcbind : ALL
```

By blocking all clients first, only clients in `/etc/hosts.allow` (added below) will be allowed to access the server.

Now add the following line to `/etc/hosts.allow`:

```
rpcbind : <NFS server IP address>
```

where `<NFS server IP address>` is the IP address of the server.

A More Complex NFS Server

NFS user permissions are based on user ID (UID). UIDs of any users on the client must match those on the server in order for the users to have access. The typical ways of doing this are:

- Manual password file synchronisation
- Use of LDAP
- Use of DNS
- Use of NIS

Note that you have to be careful on systems where the main user has root access: that user can change UIDs on the system to allow themselves access to anyone's files. This page assumes that the administrative team is the only group with root access and that

they are all trusted. Anything else represents a more advanced configuration, and will not be addressed here.

Group permissions

A user's file access is determined by their membership of groups on the client, not on the server. However, there is an important limitation: a maximum of 16 groups are passed from the client to the server, and if a user is member of more than 16 groups on the client, some files or directories might be unexpectedly inaccessible.

DNS (optional, only if using DNS)

Add any client name and IP addresses to `/etc/hosts`. (The IP address of the server should already be there.) This ensures that NFS will still work even if DNS goes down. Alternatively you can rely on DNS if you want - it's up to you.

NIS (optional, only if using NIS)

This applies to clients using NIS. Otherwise you can't use netgroups, and should specify individual IPs or hostnames in `/etc/exports`. Read the BUGS section in `man netgroup` for more information.

First, edit `/etc/netgroup` and add a line to classify your clients (this step is not necessary, but is for convenience):

```
myclients (client1,,) (client2,,) ...
```

where `myclients` is the netgroup name.

Next run this command to rebuild the NIS database:

```
sudo make -C /var/yp
```

The filename `yp` refers to Yellow Pages, the former name of NIS.

Portmap lockdown (optional)

Add the following line to `/etc/hosts.deny`:

```
rpcbind mountd nfsd statd lockd rquotad : ALL
```

By blocking all clients first, only clients in `/etc/hosts.allow` (added below) will be allowed to access the server.

Consider adding the following line to `/etc/hosts.allow`:

```
rpcbind mountd nfsd statd lockd rquotad : <list of IPs>
```

where `<list of IPs>` is a list of the IP addresses of the server and all clients. These have to be IP addresses because of a limitation in `rpcbind`. Note that if you have NIS set up, you can just add these to the same line.

Package installation and configuration

Install the necessary packages:

```
sudo apt install rpcbind nfs-kernel-server
```

Edit `/etc/exports` and add the shares:

```
/home @myclients(rw,sync,no_subtree_check)
/usr/local @myclients(rw,sync,no_subtree_check)
```

The example above shares `/home` and `/usr/local` to all clients in the `myclients` netgroup.

```
/home 192.168.0.10(rw,sync,no_subtree_check) 192.168.0.11(rw,sync,no_subtree_ch
eck)
/usr/local 192.168.0.10(rw,sync,no_subtree_check) 192.168.0.11(rw,sync,no_subtr
ee_check)
```

The example above shares `/home` and `/usr/local` to two clients with static IP addresses. If you want instead to allow access to all clients in the private network falling within a designated IP address range, consider the following:

```
/home 192.168.0.0/255.255.255.0(rw,sync,no_subtree_check)
/usr/local 192.168.0.0/255.255.255.0(rw,sync,no_subtree_check)
```

Here, `rw` makes the share read/write, and `sync` requires the server to only reply to requests once any changes have been flushed to disk. This is the safest option; `async` is faster, but dangerous. It is strongly recommended that you read `man exports` if you are considering other options.

After setting up `/etc/exports`, export the shares:

```
sudo exportfs -ra
```

You'll want to run this command whenever `/etc/exports` is modified.

Restart services

If any changes are made, rpcbind and NFS will need to be restarted:

```
sudo systemctl restart rpcbind
sudo systemctl restart nfs-kernel-server
```

Security items to consider

Aside from the UID issues discussed above, it should be noted that an attacker could potentially masquerade as a machine that is allowed to map the share, which allows them to create arbitrary UIDs to access your files. One potential solution to this is IPSec. You can set up all your domain members to talk to each other only over IPSec, which will effectively authenticate that your client is who it says it is.

IPSec works by encrypting traffic to the server with the server's public key, and the server sends back all replies encrypted with the client's public key. The traffic is decrypted with the respective private keys. If the client doesn't have the keys that it is supposed to have, it can't send or receive data.

An alternative to IPSec is physically separate networks. This requires a separate network switch and separate Ethernet cards, and physical security of that network.

Troubleshooting

Mounting an NFS share inside an encrypted home directory will only work after you are successfully logged in and your home is decrypted. This means that using /etc/fstab to mount NFS shares on boot will not work, because your home has not been decrypted at the time of mounting. There is a simple way around this using symbolic links:

1. Create an alternative directory to mount the NFS shares in:

```
sudo mkdir /nfs
sudo mkdir /nfs/music
```

1. Edit `/etc/fstab` to mount the NFS share into that directory instead:

```
nfsServer:music    /nfs/music    nfs    auto    0 0
```

1. Create a symbolic link inside your home, pointing to the actual mount location. For example, and in this case deleting the `Music` directory already existing there first:

```
rmdir /home/user/Music
ln -s /nfs/music/ /home/user/Music
```

Samba (SMB/CIFS)

Edit this on GitHub

Samba is an implementation of the SMB/CIFS networking protocol that is used by Microsoft Windows devices to provide shared access to files, printers, and serial ports.

You can use Samba to mount a folder shared from a Windows machine so it appears on your Raspberry Pi, or to share a folder from your Raspberry Pi so it can be accessed by your Windows machine.

Installing Samba Support

By default, Raspberry Pi OS does not include CIFS/Samba support, but this can be added. The following commands will install all the required components for using Samba as a server or a client.

```
sudo apt update
sudo apt install samba samba-common-bin smbclient cifs-utils
```

Mount a Folder Shared from Windows

First, you need to share a folder on your Windows device. This is quite a convoluted process!

Turn on sharing

1. Open the Networking and Sharing Centre by right-clicking on the system tray and selecting it
2. Click on **Change advanced sharing settings**
3. Select **Turn on network discovery**
4. Select **Turn on file and printer sharing**
5. Save changes

Share the folder

You can share any folder you want, but for this example, simply create a folder called `share`.

1. Create the folder `share` on your desktop.
2. Right-click on the new folder, and select **Properties**.
3. Click on the **Sharing** tab, and then the **Advanced Sharing** button
4. Select **Share this folder**; by default, the share name is the name of the folder
5. Click on the **Permissions** button
6. For this example, select **Everyone** and **Full Control** (you can limit access to specific users if required); click **OK** when done, then **OK** again to leave the **Advanced Sharing** page
7. Click on the **Security** tab, as we now need to configure the same permissions
8. Select the same settings as the **Permissions** tab, adding the chosen user if necessary
9. Click **OK**

The folder should now be shared.

Windows 10 Sharing Wizard

On Windows 10 there is a Sharing Wizard that helps with some of these steps.

1. Run the Computer Management application from the Start Bar
2. Select **Shared Folders**, then **Shares**
3. Right-click and select **New Share**, which will start up the Sharing Wizard; click **Next**
4. Select the folder you wish to share, and click **Next**
5. Click **Next** to use all the sharing defaults
6. Select **Custom** and set the required permissions, and click **OK**, then **Finish**

Mount the folder on the Raspberry Pi

Mounting in Linux is the process of attaching a folder to a location, so firstly we need that location.

```
mkdir windowshare
```

Now, we need to mount the remote folder to that location. The remote folder is the host name or IP address of the Windows PC, and the share name used when sharing it. We also need to provide the Windows username that will be used to access the remote machine.

```
sudo mount.cifs //<hostname or IP address>/share /home/pi/windowshare -o user=<
name>
```

You should now be able to view the content of the Windows share on your Raspberry Pi.

```
cd windowshare
ls
```

"Host is down" error

This error is caused by a combination of two things: A SMB protocol version mismatch, and the CIFS client on Linux returning a misleading error message. In order to fix this a version entry needs to be added to the mount command. By default Raspberry Pi OS will only use versions 2.1 and above, which are compatible with Windows 7 and later. Older devices, including some NAS, may require version 1.0:

```
sudo mount.cifs //IP/share /mnt/point -o user=<uname>,vers=1.0
```

You may need to try different versions to match up with the server version. Possible values are:

Version	Description
1.0	Classic CIFS/SMBv1 protocol
2.0	The SMBv2.002 protocol. Windows Vista Service Pack 1, and Windows Server 2008
2.1	The SMBv2.1 protocol. Microsoft Windows 7 and Windows Server 2008R2
3.0	The SMBv3.0 protocol. Microsoft Windows 8 and Windows Server 2012
3.02	The SMBv3.0.2 protocol. Microsoft Windows 8.1 and Windows Server 2012R2
3.11	The SMBv3.1.1 protocol. Microsoft Windows 10 and Windows Server 2016
3	The SMBv3.0 protocol version and above

Sharing a Folder from your Raspberry Pi

Firstly, create a folder to share. This example creates a folder called `shared` in the `home` folder of the current user:

```
$ cd ~
$ mkdir shared
$ chmod 0740 shared
```

Now we need to tell Samba about your default user account when accessing that folder. When prompted, enter your password, replacing the `<username>` placeholder with the username of your primary user account:

```
$ sudo smbpasswd -a <username>
```

Now we need to tell Samba to share this folder, using the Samba configuration file.

```
sudo nano /etc/samba/smb.conf
```

At the end of the file, add the following to share the folder, giving the remote user read/write permissions. Replace the `<username>` placeholder with the username of your primary user account:

```
[share]
    path = /home/<username>/shared
    read only = no
    public = yes
    writable = yes
```

In the same file, find the `workgroup` line, and if necessary, change it to the name of the workgroup of your local Windows network.

```
workgroup = <your workgroup name here>
```

That should be enough to share the folder. On your Windows device, when you browse the network, the folder should appear and you should be able to connect to it.

Virtual Network Computing (VNC)

Edit this on GitHub

Sometimes it is not convenient to physically work with a device. Virtual Network Computing (VNC) allows you to control the desktop of one device from another.

VNC relies upon a client and a server. The client runs on a device you can physically interact with, such as a personal laptop, desktop, tablet, or phone. The server runs on your

Raspberry Pi. When you use VNC, the client transmits keyboard and mouse events to the server. The server executes those events on your Raspberry Pi, and returns screen updates to the client.
The VNC client displays the desktop of your Raspberry Pi in a window. You can interact with the desktop as though you were working on the Raspberry Pi itself.

Raspberry Pi OS includes wayvnc. This provides a VNC server that you can enable in your device preferences.

Before you can use VNC on your Raspberry Pi, you must enable the VNC server.

NOTE

Previous versions of Raspberry Pi OS supported client connections from RealVNC. In Raspberry Pi OS *Bookworm* or later, RealVNC is no longer supported. Instead, you can connect using TigerVNC.

Enable the VNC server

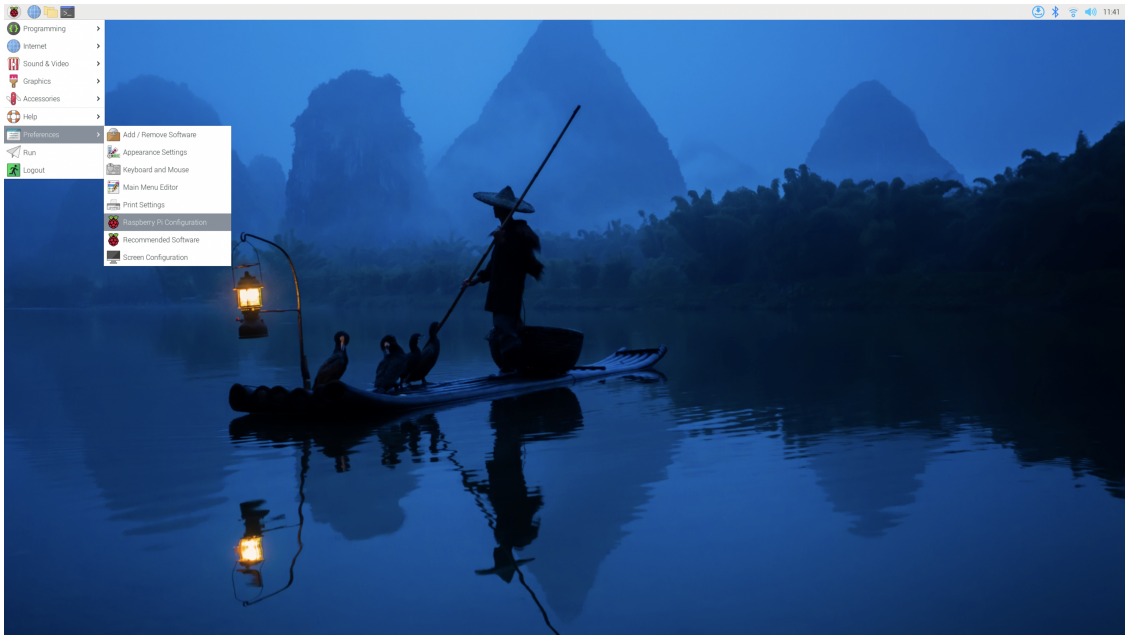
Raspberry Pi OS supports enabling the VNC server both graphically and at the command line.

TIP

Once enabled, you can access your WayVNC configuration at `~/.config/wayvnc/`.

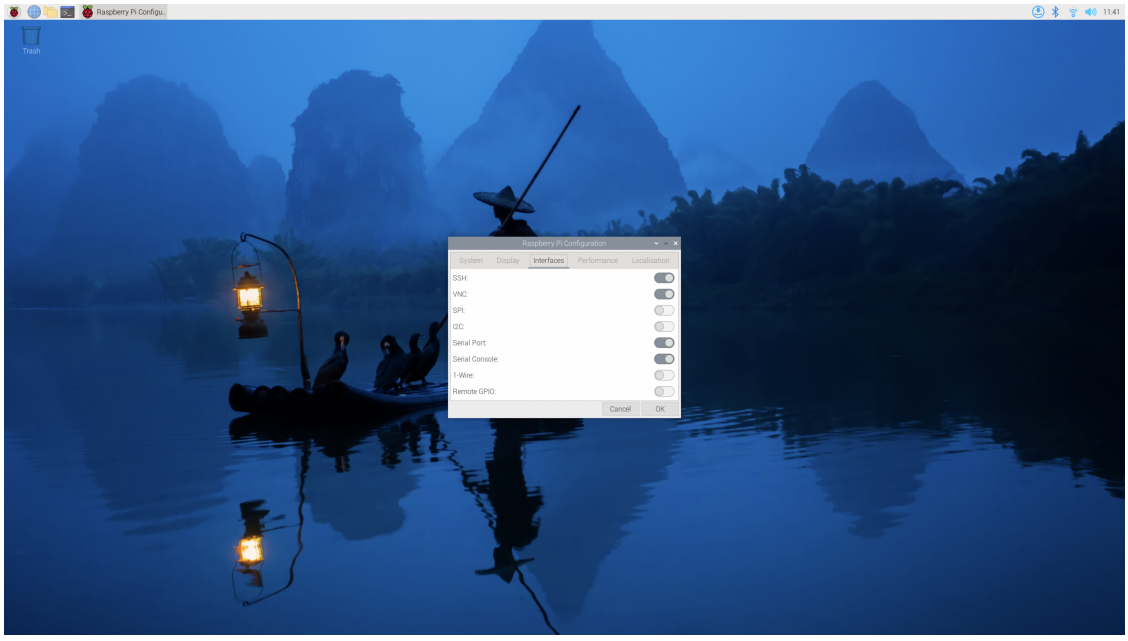
Enable VNC Server Graphically

1. Boot into the graphical desktop on your Raspberry Pi.
2. Click the Raspberry Pi icon in the system tray of your desktop.
3. Select **Preferences** > **Raspberry Pi Configuration** from the menu.



4. Navigate to the **Interfaces** tab.

5. Click the radio button next to **VNC** into the active position.



6. Click the "OK" button to save your configuration changes.

Enable the VNC server on the command line

Use using raspi-config to enable the VNC server on the command line.

1. Open `raspi-config` with the following line:

```
sudo raspi-config
```

2. Navigate to **Interface Options**. Press `Enter` to select.

3. Select **VNC**. Press `Enter` to select.

4. Under "Would you like the VNC Server to be enabled?", highlight `<Yes>` and press `Enter`.

5. Press `Enter` to return to the menu. Press `Esc` to exit `raspi-config`.

Connect to your Raspberry Pi

To connect to your Raspberry Pi, you'll need the following:

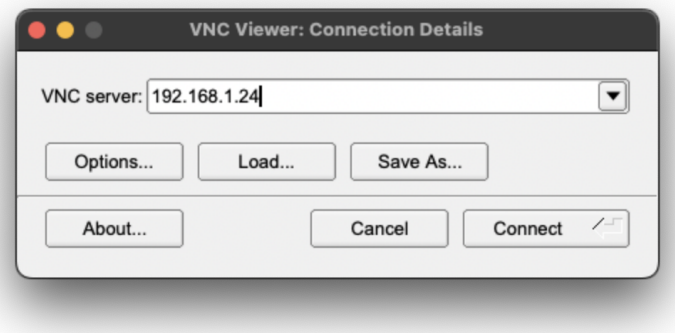
- your Raspberry Pi and the device running the VNC client must be connected to the same network (e.g. a home wireless network)
- the IP address of your Raspberry Pi
- a valid username and password combination for an account on your Raspberry Pi

If you don't know the IP address of your device, see our instructions on finding your IP address.

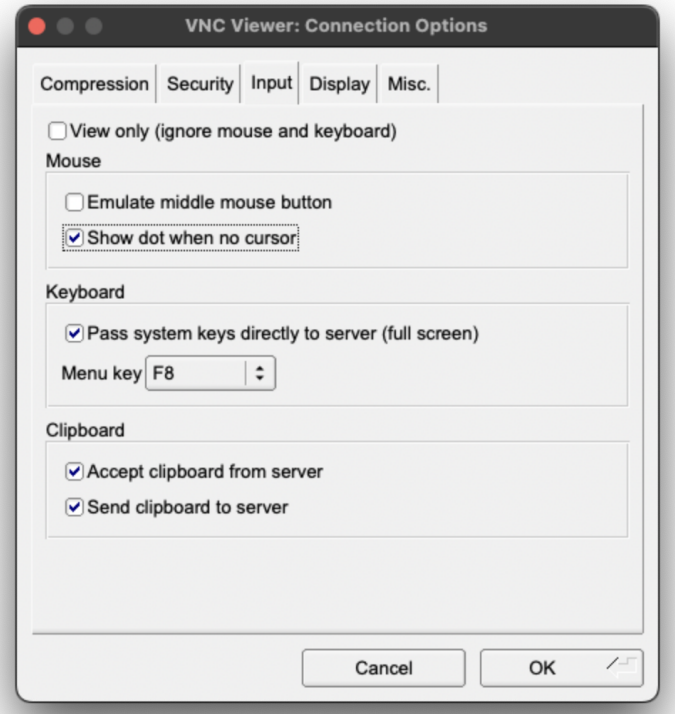
1. Download TigerVNC. You can install the latest version from the Releases page of their GitHub repository. Click on the link in the latest release, and find the binary for your platform. Windows users should download an `exe`; macOS users should download the `dmg`; Linux users should install the `jar`.

2. On your client device, launch TigerVNC. On macOS and Windows, you can double-click the binary. On Linux, install java with `sudo apt install default-jre`, then run `java -jar VncViewer-<version>.jar`, replacing the `<version>` placeholder with the version you downloaded.

3. In the "VNC server" field, enter the IP address of your Raspberry Pi.

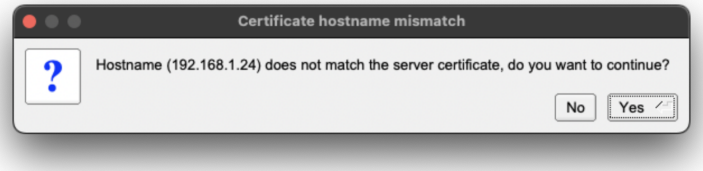


4. Click the "Options" button. Navigate to the "Input" tab. Check the box next to "Show dot when no cursor" to ensure that you can always see a cursor in TigerVNC.

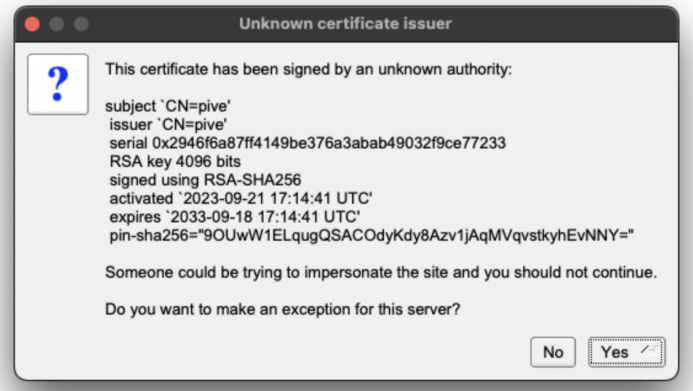


5. Click the "Connect" button to initiate a connection with the server.

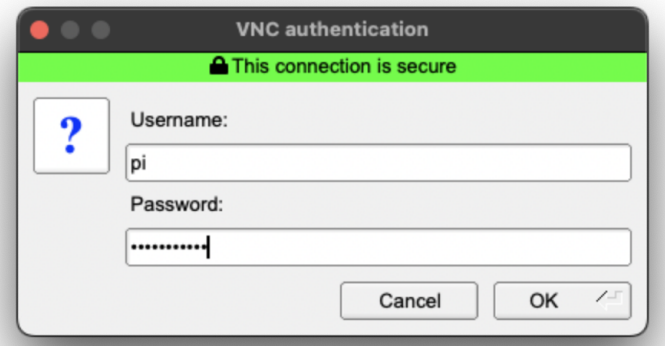
    - If TigerVNC warns you that the "Hostname does not match the server certificate", click the "Yes" button to continue.



    - If TigerVNC warns you that the "certificate has been signed by an unknown authority", click the "Yes" button to grant an exception for your Raspberry Pi.



6. When prompted for a username and password, enter your credentials.



7. Click the "OK" button to authenticate with the VNC server. If your credentials are correct, TigerVNC should open a window containing the desktop corresponding to your account on the Raspberry Pi. You should be able to move your mouse and keyboard to input text and interact with the desktop.

Setting up an Apache Web Server

Edit this on GitHub

Apache is a popular web server application you can install on the Raspberry Pi to allow it to serve web pages.

On its own, Apache can serve HTML files over HTTP, and with additional modules can serve dynamic web pages using scripting languages such as PHP.

Installing Apache

First, update the available packages by typing the following command into the Terminal:

```
sudo apt update
```

Then, install the `apache2` package with this command:

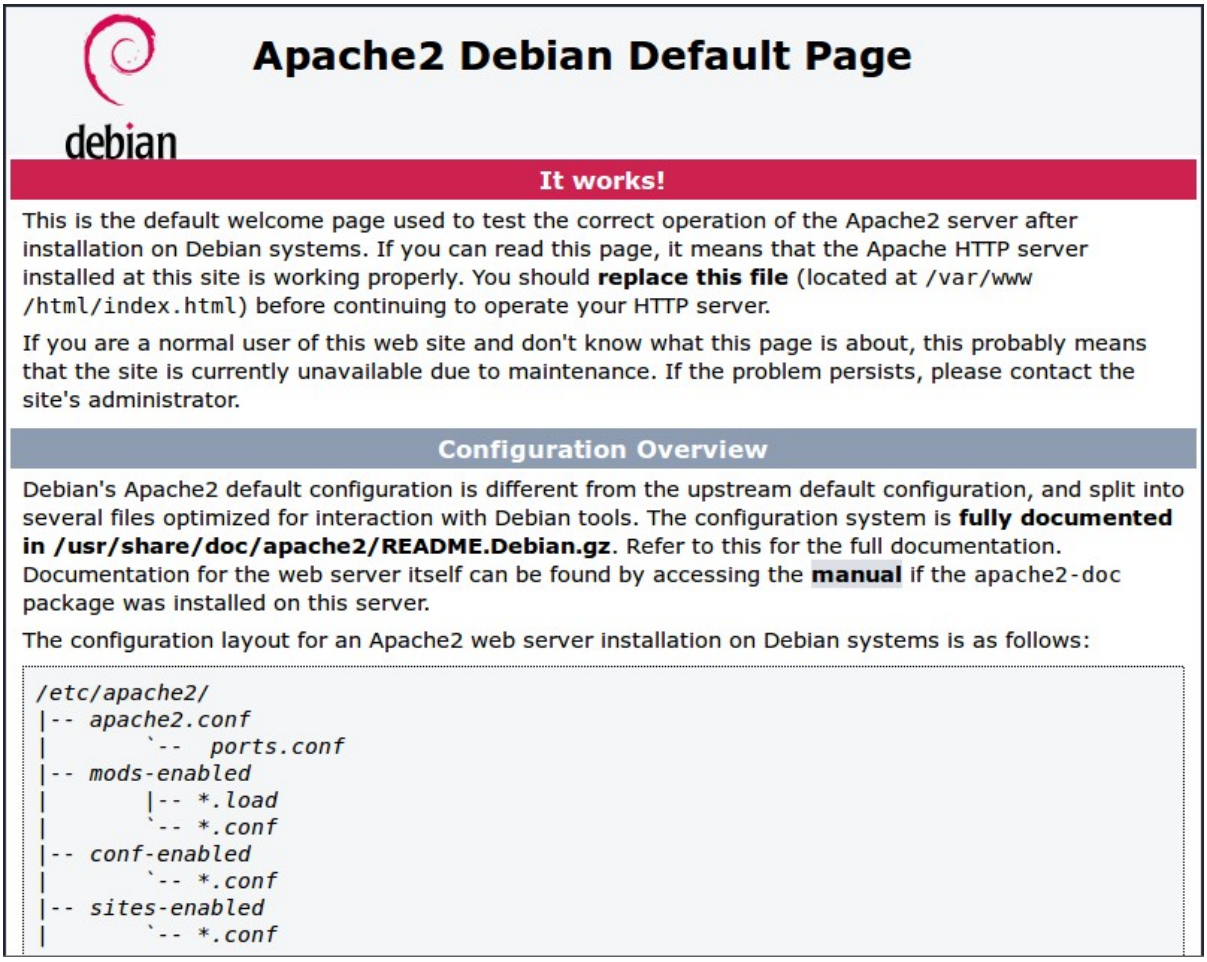
```
sudo apt install apache2 -y
```

Test the Web Server

By default, Apache puts a test HTML file in the web folder. This default web page is served when you browse to `http://localhost/` on the Raspberry Pi itself, or `http://192.168.1.10` (whatever the Raspberry Pi's IP address is) from another computer on the network. To find

the Raspberry Pi's IP address, type `hostname -I` at the command line (or read more about finding your IP address).

Browse to the default web page either on the Raspberry Pi or from another computer on the network and you should see the following:



This means you have Apache working!

Changing the Default Web Page

This default web page is just an HTML file on the filesystem. It is located at `/var/www/html/index.html`.

Navigate to this directory in a terminal window and have a look at what's inside:

```
cd /var/www/html
ls -al
```

This will show you:

```
total 12
drwxr-xr-x  2 root root 4096 Jan  8 01:29 .
```

```
drwxr-xr-x 12 root root 4096 Jan  8 01:28 ..
-rw-r--r--  1 root root  177 Jan  8 01:29 index.html
```

This shows that by default there is one file in `/var/www/html/` called `index.html` and it is owned by the `root` user (as is the enclosing folder). In order to edit the file, you need to change its ownership to your own username. Change the owner of the file using the following command, replacing the `<username>` placeholder with the username of your primary user account:

```
$ sudo chown <username>: index.html
```

You can now try editing this file and then refreshing the browser to see the web page change. If you know HTML you can put your own HTML files and other assets in this directory and serve them as a website on your local network.

Installing PHP for Apache

To allow your Apache server to process PHP files, you'll need to install the latest version of PHP and the PHP module for Apache. Type the following command to install these:

```
sudo apt install php libapache2-mod-php -y
```

Now remove the `index.html` file:

```
sudo rm index.html
```

and create the file `index.php`:

```
sudo nano index.php
```

Put some PHP content in it:

```
<?php echo "hello world"; ?>
```

Now save and refresh your browser. You should see "hello world". This is not dynamic but still served by PHP. Try something dynamic:

```
<?php echo date('Y-m-d H:i:s'); ?>
```

or show your PHP info:

```
<?php phpinfo(); ?>
```

Network boot your Raspberry Pi

Edit this on GitHub

You can set up a DHCP/TFTP server which will allow you to boot a Raspberry Pi 3 or 4 from the network.

The instructions assume that you have an existing home network, and that you want to use a Raspberry Pi for the **server**. You will also need an additional Raspberry Pi 3 or 4 as a **client** to be booted. Only one SD Card is needed because the client will be booted from the server after the initial client configuration.

NOTE

Due to the huge range of networking devices and routers available, we can't guarantee that network booting will work with any device. We have had reports that, if you cannot get network booting to work, disabling STP frames on your network may help.

Client Configuration

Raspberry Pi 3 Model B

NOTE

This section only applies to the Raspberry Pi 3 Model B, as network boot is enabled on the Raspberry Pi 3 Model B+ at the factory.

Before the Raspberry Pi 3 Model B will network boot it needs to be booted from an SD Card with a config option to enable USB boot mode. This will set a bit in the OTP (One Time Programmable) memory in the Raspberry Pi SoC that enables network booting. Once this is done, the Raspberry Pi 3B will attempt to boot from USB, and from the network, if it cannot boot from the SD card.

Install Raspberry Pi OS Lite, or Raspberry Pi OS with desktop, on the SD card in the usual fashion. Next, enable USB boot mode with the following command:

```
echo program_usb_boot_mode=1 | sudo tee -a /boot/firmware/config.txt
```

This adds `program_usb_boot_mode=1` to the end of `/boot/firmware/config.txt`. Reboot the Raspberry Pi with `sudo reboot`. Once the client Raspberry Pi has rebooted, check that the OTP has been programmed with:

```
vcgencmd otp_dump | grep 17:
17:3020000a
```

Ensure the output `0x3020000a` is correct.

The client configuration is almost done. As a final step, disable USB booting. Run the following command:

```
sudo nano /boot/firmware/config.txt
```

Remove the line that contains the text `program_usb_boot_mode=1`. Finally, shut the client Raspberry Pi down with `sudo poweroff`.

Raspberry Pi 4 Model B

Network boot can be enabled on the Raspberry Pi 4 using the `raspi-config` tool. First, run `raspi-config` as follows:

```
sudo raspi-config
```

Within `raspi-config`, choose `Advanced Options`, then `Boot Order`, then `Network Boot`. You must then reboot the device for the change to the boot order to be programmed into the bootloader EEPROM. Once the Raspberry Pi has rebooted, check that the boot order is now `0xf21`:

```
vcgencmd bootloader_config
```

For further details of configuring the Raspberry Pi 4 bootloader, see Raspberry Pi Bootloader Configuration.

Ethernet MAC address

Before configuring network boot, make a note of the serial number and mac address so that the board can be identified by the TFTP/DHCP server.

On Raspberry Pi 4 the MAC address is programmed at manufacture and there is no link between the MAC address and serial number. Both the MAC address and serial numbers are displayed on the bootloader HDMI diagnostics screen.

To find the Ethernet MAC address:

```
ethtool -P eth0
```

To find the serial number:

```
grep Serial /proc/cpuinfo | cut -d ' ' -f 2 | cut -c 9-16
```

Server Configuration

Plug the SD card into the server Raspberry Pi, and then boot the server. The client Raspberry Pi will need a root file system to boot from: we will use a copy of the server's root filesystem and place it in `/nfs/client1`:

```
sudo mkdir -p /nfs/client1
sudo apt install rsync
sudo rsync -xa --progress --exclude /nfs / /nfs/client1
```

Regenerate SSH host keys on the client filesystem by chrooting into it:

```
cd /nfs/client1
sudo mount --bind /dev dev
sudo mount --bind /sys sys
sudo mount --bind /proc proc
sudo chroot .
rm /etc/ssh/ssh_host_*
dpkg-reconfigure openssh-server
exit
sudo umount dev sys proc
```

Find the settings of your local network. You need to find the address of your router (or gateway), which can be done with:

```
ip route | awk '/default/ {print $3}'
```

Then run:

```
ip -4 addr show dev eth0 | grep inet
```

which should give an output like:

```
inet 10.42.0.211/24 brd 10.42.0.255 scope global eth0
```

The first address is the IP address of your server Raspberry Pi on the network, and the part after the slash is the network size. It is highly likely that yours will be a `/24`. Also note the `brd` (broadcast) address of the network. Note down the output of the previous command, which will contain the IP address of the Raspberry Pi and the broadcast address of the network.

Finally, note down the address of your DNS server, which is the same address as your gateway. You can find this with:

```
cat /etc/resolv.conf
```

Configure a static network address on your server Raspberry Pi via the `systemd` networking, which works as the network handler and DHCP server.

To do that, you'll need to create a `10-eth0.netdev` and a `11-eth0.network` like so:

```
sudo nano /etc/systemd/network/10-eth0.netdev
```

Add the following lines:

```
[Match]
Name=eth0

[Network]
DHCP=no
```

Then create a network file:

```
sudo nano /etc/systemd/network/11-eth0.network
```

Add the following contents:

```
[Match]
Name=eth0

[Network]
Address=10.42.0.211/24
DNS=10.42.0.1

[Route]
Gateway=10.42.0.1
```

At this point, you will not have working DNS, so you will need to add the server you noted down before to `systemd/resolved.conf`. In this example, the gateway address is 10.42.0.1.

```
sudo nano /etc/systemd/resolved.conf
```

Uncomment the DNS line and add the DNS IP address there. Additionally, if you have a fallback DNS server, add it there as well.

```
[Resolve]
DNS=10.42.0.1
#FallbackDNS=
```

Enable `systemd-networkd` and then reboot for the changes to take effect:

```
sudo systemctl enable systemd-networkd
sudo reboot
```

Now start `tcpdump` so you can search for DHCP packets from the client Raspberry Pi:

```
sudo apt install tcpdump dnsmasq
sudo systemctl enable dnsmasq
sudo tcpdump -i eth0 port bootpc
```

Connect the client Raspberry Pi to your network and power it on. Check that the LEDs illuminate on the client after around 10 seconds, then you should get a packet from the client "DHCP/BOOTP, Request from …"

```
IP 0.0.0.0.bootpc > 255.255.255.255.bootps: BOOTP/DHCP, Request from b8:27:e
b...
```

Now you need to modify the `dnsmasq` configuration to enable DHCP to reply to the device. Press `CTRL + C` to exit the `tcpdump` program, then type the following:

```
echo | sudo tee /etc/dnsmasq.conf
sudo nano /etc/dnsmasq.conf
```

Then replace the contents of `dnsmasq.conf` with:

```
# Note: comment out port if you want DNS services for systems on the network.
port=0
dhcp-range=10.42.0.255,proxy
log-dhcp
enable-tftp
tftp-root=/tftpboot
pxe-service=0,"Raspberry Pi Boot"
```

Where the first address of the `dhcp-range` line is, use the broadcast address you noted down earlier.

Now create a `/tftpboot` directory:

```
sudo mkdir /tftpboot
sudo chmod 777 /tftpboot
sudo systemctl enable dnsmasq.service
sudo systemctl restart dnsmasq.service
```

Now monitor the `dnsmasq` log:

```
journalctl -f
```

You should see something like this:

```
raspberrypi dnsmasq-tftp[1903]: file /tftpboot/bootcode.bin not found
```

Next, you will need to copy the contents of the boot folder into the `/tftpboot` directory.

First, press `CTRL + C` to exit the monitoring state. Then type the following:

```
cp -r /boot/firmware/* /tftpboot
```

Since the tftp location has changed, restart `dnsmasq`:

```
sudo systemctl restart dnsmasq
```

Set up NFS root

This should now allow your Raspberry Pi client to attempt to boot through until it tries to load a root file system (which it doesn't have).

At this point, export the `/nfs/client1` file system created earlier, and the TFTP boot folder.

```
sudo apt install nfs-kernel-server
echo "/nfs/client1 *(rw,sync,no_subtree_check,no_root_squash)" | sudo tee -a /e
tc/exports
echo "/tftpboot *(rw,sync,no_subtree_check,no_root_squash)" | sudo tee -a /etc/
exports
```

Restart RPC-Bind and the NFS server in order to have them detect the new files.

```
sudo systemctl enable rpcbind
sudo systemctl restart rpcbind
sudo systemctl enable nfs-kernel-server
sudo systemctl restart nfs-kernel-server
```

Edit `/tftpboot/cmdline.txt` and from `root=` onwards, and replace it with:

```
root=/dev/nfs nfsroot=10.42.0.211:/nfs/client1,vers=3 rw ip=dhcp rootwait
```

You should substitute the IP address here with the IP address you have noted down. Also remove any part of the command line starting with init=.

Finally, edit `/nfs/client1/etc/fstab` and remove the `/dev/mmcblk0p1` and `p2` lines (only `proc` should be left). Then, add the boot partition back in:

```
echo "10.42.0.211:/tftpboot /boot/firmware/ nfs defaults,vers=3 0 0" | sudo tee
-a /nfs/client1/etc/fstab
```

Good luck! If it doesn't boot on the first attempt, keep trying. It can take a minute or so for the Raspberry Pi to boot, so be patient.

Using `pxetools`

We have created a Python script that is used internally to quickly set up Raspberry Pis that will network boot.

The script takes a serial number, which you can find in `cat /proc/cpuinfo`, an owner name and the name of the Raspberry Pi. It then creates a root filesystem for that Raspberry Pi from a Raspberry Pi OS image. There is also a `--list` option which will print out the IP address of the Raspberry Pi, and a `--remove` option.

NOTE

> The following instructions describe how to set up the environment required by the script starting from a fresh Raspberry Pi OS lite image. It might be a good idea to mount a hard disk or flash drive on `/nfs` so that your SD card isn't providing filesystems to multiple Raspberry Pis. This is left as an exercise for the reader.

```
sudo apt update
sudo apt full-upgrade -y
sudo reboot

wget https://datasheets.raspberrypi.com/soft/prepare_pxetools.sh
bash prepare_pxetools
```

When prompted about saving `iptables` rules, say `no`.

The `prepare_pxetools` script should prepare everything you need to use `pxetools`.

We found that we needed to restart the `nfs` server after using `pxetools` for the first time. Do this with:

```
sudo systemctl restart nfs-kernel-server
```

Then plug in your Raspberry Pi and it should boot!

Network booting using IPv6

Edit this on GitHub

There are 4 stages to booting a Raspberry Pi computer over the network:

1. The bootloader negotiates to get an IP address and the details of a TFTP server using DHCP.
2. The bootloader loads the firmware via TFTP and hands over the boot process to the firmware, passing it the details of the network.
3. The firmware loads the kernel and command line via TFTP.
4. The kernel boots the rest of the system, loading the root filesystem (rootfs) via NFS or some other mechanism.

The bootloader and firmware (stages 1 to 3) have been enhanced to support booting over IPv6.

IMPORTANT

IPv6 netboot is an **experimental alpha feature** and depending on feedback, we may need to change how it works in future. This only works on Raspberry Pi 4 and Compute Module 4.

How it works

To boot via IPv6 you need an updated version of the firmware (e.g. `start4.elf`) and the bootloader. Using the latest release of Raspberry Pi OS and the latest stable bootloader should be sufficient.

NOTE

The commonly used `dnsmasq` DHCP server doesn't currently support the network boot parameters required for IPv6 network boot, so for the time being you will have to use a different DHCP server such as ISC DHCP.

To mount `rootfs` over the network the IPv4 netboot tutorial suggests using `nfsroot`. This doesn't support IPv6, so another method is needed to mount `rootfs` over the network.

If your ISP and router don't support IPv6 you will be limited in what you can do.

Network addresses

The first thing the bootloader does is send a router solicitation to get the details of the network. The router responds with an advertisement packet identifying its ethernet address, which the bootloader might need if the TFTP server is on a different network.

The router advertisement includes a flag which tells it whether to use stateful (managed) or stateless (unmanaged) configuration for its IP address. Stateless configuration means that the device configures its own IP address. Currently the bootloader generates an address derived from its ethernet MAC address and a network prefix supplied by the router.

If the router indicates that stateful configuration is enabled DHCP is used to obtain the IP address of the device. This involves the device sending a solicitation request to a DHCP server which responds with an advertisement. The client then requests the address before getting a reply acknowledgement from the server.

DHCP Servers and clients identify themselves with variable length DUID (Device Unique ID). On the Raspberry Pi this is derived from the MAC address (DUID_LL).

TFTP address

Whether using stateless or stateful configuration, the DHCP server is used to obtain the TFTP server address. This is encoded in the `BOOTFILE-URL` parameter. We send the client architecture type value `0x29` to identify a device.

See RFC 5970 and the IANA Dynamic Host Configuration Protocol for IPv6 documentation.

Boot process

The device should now have an IP address and TFTP details. It downloads the firmware binary `start4.elf` from the TFTP server and continues running with this. The firmware is passed the IP address and TFTP server details so it can download the kernel and boot the rest of the system.

Kernel Boot

With IPv4 netboot, `nfsroot` is used to mount `rootfs` over the network. This doesn't support IPv6 so another solution is required. It might involve a small RAM file system that can mount the appropriate network location before switching to the proper `rootfs` contents.

NOTE

A mechanism to boot the Linux kernel with NFS via IPv6 is still to be demonstrated.

Test Setup

If you want to try this out you will need another Raspberry Pi to act as the TFTP and DHCP server.

The TFTP server can in theory be on any routable network but the DHCP server has to be on the same network as the devices it will serve.

TFTP Server

If you have a working IPv4 network boot setup you can reuse the TFTP server in dnsmasq to supply the files (it can talk to both IPv4 and IPv6).

Alternatively you can use a standalone TFTP server like `tftpd-hpa`.

```
$ sudo apt-get install tftpd-hpa
$ sudo systemctl start tftpd-hpa
```

DHCP Server

DHCP in IPv6 has changed a lot. We need DHCP to at least tell us the address of the TFTP server, which in this case is the same machine.

```
$ sudo apt-get install isc-dhcp-server
```

Modify the configuration in `/etc/default/isc-dhcp-server`

```
DHCPDv6_CONF=/etc/dhcp/dhcpd6.conf
INTERFACESv6="eth0"
```

In `/etc/dhcp/dhcpd6.conf` you need to specify the TFTP server address and setup a subnet. Here the DHCP server is configured to supply some made up unique local addresses (ULA). The `host test-rpi4` line tells DHCP to give a test device a fixed address.

```
not authoritative;

# Check if the client looks like a Raspberry Pi
if option dhcp6.client-arch-type = 00:29 {
        option dhcp6.bootfile-url "tftp://[fd49:869:6f93::1]/";
}
```

```
subnet6 fd49:869:6f93::/64 {
        host test-rpi4 {
                host-identifier option dhcp6.client-id 00:03:00:01:e4:5f:01:20:
24:0b;
                fixed-address6 fd49:869:6f93::1000;
        }
}
```

Your server has to be assigned the IPv6 address in `/etc/dhcpcd.conf`

```
interface eth0
static ip6_address=fd49:869:6f93::1/64
```

Now start the DHCP server.

```
$ sudo systemctl restart isc-dhcp-server.service
```

Bootloader

Modify the configuration to tell it to attempt network boot via IPv6 rather than IPv4.

```
BOOT_ORDER=0xf21 # 2=Network boot
USE_IPV6=1 # Enable IPv6 network boot
BOOT_UART=1 # Debug
```

To revert to IPv4 network boot just remove the `USE_IPV6` line from `boot.conf`.

Router

To use IPv6 you really need a router and ISP that supports IPv6. There are sites on the internet that can check this for you or alternatively run the following command.

```
sudo apt-get install ndisc6
rdisc6 -1 eth0
```

This sends a router solicitation to your router asking for your network details such as the network prefix, router ethernet address and whether to use DHCP for addressing. If there's no response to this command it's likely your network and ISP only supports IPv4. If IPv6 is supported it's most likely that it will be configured to use stateless configuration where clients generate their own addresses.

```
Soliciting ff02::2 (ff02::2) on eth0...
Hop limit                 :           64 (      0x40)
Stateful address conf.    :           No
Stateful other conf.      :          Yes
Mobile home agent         :           No
```

```
Router preference          :       medium
Neighbor discovery proxy   :           No
Router lifetime            :          180 (0x000000b4) seconds
Reachable time             :  unspecified (0x00000000)
Retransmit time            :  unspecified (0x00000000)
```

You might be able to configure your router for stateful configuration, which means it will use DHCP to obtain an IP address.

```
Hop limit                  :           64 (      0x40)
Stateful address conf.     :          Yes
Stateful other conf.       :          Yes
Mobile home agent          :           No
Router preference          :       medium
Neighbor discovery proxy   :           No
Router lifetime            :          180 (0x000000b4) seconds
Reachable time             :  unspecified (0x00000000)
Retransmit time            :  unspecified (0x00000000)
```

Debugging

Logs and Traces

If the boot uart is enabled you should see something like this from the serial port. The lines starting RX6 indicate that IPv6 is in use.

Here `dc:a6:32:6f:73:f4` is the MAC address of the TFTP server and it has an IPv6 address of `fd49:869:6f93::1`. The device itself has a MAC address `e4:5f:01:20:24:0b` and an IPv6 address of `fd49:869:6f93::1000`

```
Boot mode: NETWORK (02) order f
GENET: RESET_PHY
PHY ID 600d 84a2
NET_BOOT: e4:5f:01:20:24:0b wait for link TFTP6: (null)
LINK STATUS: speed: 100 full duplex
Link ready
GENET START: 64 16 32
GENET: UMAC_START 0xe45f0120 0x240b0000
RX6: 12 IP: 1 MAC: 1 ICMP: 1/1 UDP: 0/0 ICMP_CSUM_ERR: 0 UDP_CSUM_ERR: 0
NET fd49:869:6f93::1000 tftp fd49:869:6f93::1
RX6: 17 IP: 4 MAC: 4 ICMP: 2/2 UDP: 2/2 ICMP_CSUM_ERR: 0 UDP_CSUM_ERR: 0
TFTP_GET: dc:a6:32:6f:73:f4 fd49:869:6f93::1 ab5a4158/start4.elf

RX6: 17 IP: 4 MAC: 4 ICMP: 2/2 UDP: 2/2 ICMP_CSUM_ERR: 0 UDP_CSUM_ERR: 0
RX6: 18 IP: 5 MAC: 5 ICMP: 2/2 UDP: 3/3 ICMP_CSUM_ERR: 0 UDP_CSUM_ERR: 0
TFTP_GET: dc:a6:32:6f:73:f4 fd49:869:6f93::1 ab5a4158/config.txt
```

Finally the bootloader hands over to firmware which should load the kernel.

Stateful configuration

You can examine network activity with tcpdump.

```
$ sudo tcpdump -i eth0 -e ip6 -XX -l -v -vv
```

Below is an extract of a TCP dump where the router is configured to use stateful (DHCP) network configuration.

Device sends a router solicitation.

```
12:23:35.387046 e4:5f:01:20:24:0b (oui Unknown) > 33:33:00:00:00:02 (oui Unknow
n), ethertype IPv6 (0x86dd), length 70: (hlim 255, next-header ICMPv6 (58) payl
oad length: 16) fe80::e65f:1ff:fe20:240b > ip6-allrouters: [icmp6 sum ok] ICMP
6, router solicitation, length 16
          source link-address option (1), length 8 (1): e4:5f:01:20:24:0b
            0x0000:  e45f 0120 240b
```

Router sends a response telling the device to use stateful configuration.

```
12:23:35.498902 60:8d:26:a7:c1:88 (oui Unknown) > 33:33:00:00:00:01 (oui Unknow
n), ethertype IPv6 (0x86dd), length 110: (hlim 255, next-header ICMPv6 (58) pay
load length: 56) bthub.home > ip6-allnodes: [icmp6 sum ok] ICMP6, router advert
isement, length 56
        hop limit 64, Flags [managed, other stateful], pref medium, router life
time 180s, reachable time 0ms, retrans timer 0ms
          rdnss option (25), length 24 (3):  lifetime 60s, addr: bthub.home
            0x0000:  0000 0000 003c fe80 0000 0000 0000 628d
            0x0010:  26ff fea7 c188
          mtu option (5), length 8 (1):  1492
            0x0000:  0000 0000 05d4
          source link-address option (1), length 8 (1): 60:8d:26:a7:c1:88
            0x0000:  608d 26a7 c188
```

Device sends a DHCP solicitation.

```
12:23:35.502517 e4:5f:01:20:24:0b (oui Unknown) > 33:33:00:01:00:02 (oui Unknow
n), ethertype IPv6 (0x86dd), length 114: (hlim 255, next-header UDP (17) payloa
d length: 60) fe80::e65f:1ff:fe20:240b.dhcpv6-client > ff02::1:2.dhcpv6-server:
[udp sum ok] dhcp6 solicit (xid=8cdd56 (client-ID hwaddr type 1 e45f0120240b)
(IA_NA IAID:0 T1:0 T2:0) (option-request opt_59) (opt_61) (elapsed-time 0))
```

The DHCP server replies with an advertisement.

```
12:23:35.510478 dc:a6:32:6f:73:f4 (oui Unknown) > e4:5f:01:20:24:0b (oui Unknow
n), ethertype IPv6 (0x86dd), length 172: (flowlabel 0xad54d, hlim 64, next-head
er UDP (17) payload length: 118) fe80::537a:52c:c647:b184.dhcpv6-server > fe8
0::e65f:1ff:fe20:240b.dhcpv6-client: [bad udp cksum 0xd886 -> 0x6d26!] dhcp6 ad
vertise (xid=8cdd56 (IA_NA IAID:0 T1:3600 T2:7200 (IA_ADDR fd49:869:6f93::1000
pltime:604800 vltime:2592000)) (client-ID hwaddr type 1 e45f0120240b) (server-I
D hwaddr/time type 1 time 671211709 dca6326f73f4) (opt_59))
```

The device sends a request for an address and TFTP details to the DHCP server.

```
12:23:35.510763 e4:5f:01:20:24:0b (oui Unknown) > 33:33:00:01:00:02 (oui Unknow
n), ethertype IPv6 (0x86dd), length 132: (hlim 255, next-header UDP (17) payloa
d length: 78) fe80::e65f:1ff:fe20:240b.dhcpv6-client > ff02::1:2.dhcpv6-server:
[udp sum ok] dhcp6 request (xid=8cdd56 (client-ID hwaddr type 1 e45f0120240b)
(server-ID hwaddr/time type 1 time 671211709 dca6326f73f4) (IA_NA IAID:0 T1:0 T
2:0) (option-request opt_59) (opt_61) (elapsed-time 1))
```

The DHCP server replies, `opt_59` is used to pass the address of the TFTP server.

```
12:23:35.512122 dc:a6:32:6f:73:f4 (oui Unknown) > e4:5f:01:20:24:0b (oui Unknow
n), ethertype IPv6 (0x86dd), length 172: (flowlabel 0xad54d, hlim 64, next-head
er UDP (17) payload length: 118) fe80::537a:52c:c647:b184.dhcpv6-server > fe8
0::e65f:1ff:fe20:240b.dhcpv6-client: [bad udp cksum 0xd886 -> 0x6826!] dhcp6 re
ply (xid=8cdd56 (IA_NA IAID:0 T1:3600 T2:7200 (IA_ADDR fd49:869:6f93::1000 plti
me:604800 vltime:2592000)) (client-ID hwaddr type 1 e45f0120240b) (server-ID hw
addr/time type 1 time 671211709 dca6326f73f4) (opt_59))
```

The device sends a neighbour solicitation to the FTP server because it needs its MAC address.

```
12:23:36.510768 e4:5f:01:20:24:0b (oui Unknown) > 33:33:ff:00:00:01 (oui Unknow
n), ethertype IPv6 (0x86dd), length 86: (hlim 255, next-header ICMPv6 (58) payl
oad length: 32) fe80::e65f:1ff:fe20:240b > ff02::1:ff00:1: [icmp6 sum ok] ICMP
6, neighbor solicitation, length 32, who has fd49:869:6f93::1
          source link-address option (1), length 8 (1): e4:5f:01:20:24:0b
            0x0000:  e45f 0120 240b
```

The FTP server replies with its MAC address.

```
12:23:36.510854 dc:a6:32:6f:73:f4 (oui Unknown) > e4:5f:01:20:24:0b (oui Unknow
n), ethertype IPv6 (0x86dd), length 86: (hlim 255, next-header ICMPv6 (58) payl
oad length: 32) fd49:869:6f93::1 > fe80::e65f:1ff:fe20:240b: [icmp6 sum ok] ICM
P6, neighbor advertisement, length 32, tgt is fd49:869:6f93::1, Flags [solicite
d, override]
          destination link-address option (2), length 8 (1): dc:a6:32:6f:73:f4
            0x0000:  dca6 326f 73f4
```

TFTP requests are made by the device which should now boot over the network.

```
12:23:36.530820 e4:5f:01:20:24:0b (oui Unknown) > dc:a6:32:6f:73:f4 (oui Unknow
n), ethertype IPv6 (0x86dd), length 111: (hlim 255, next-header UDP (17) payloa
d length: 57) fd49:869:6f93::1000.61785 > fd49:869:6f93::1.tftp: [udp sum ok]
49 RRQ "ab5a4158/start4.elf" octet tsize 0 blksize 1024
```

Stateless configuration

Below is an extract of a tcp dump for a stateless (non-DHCP) network configuration.

The device sends a router solicitation.

```
12:55:27.541909 e4:5f:01:20:24:0b (oui Unknown) > 33:33:00:00:00:02 (oui Unknow
n), ethertype IPv6 (0x86dd), length 70: (hlim 255, next-header ICMPv6 (58) payl
oad length: 16) fe80::e65f:1ff:fe20:240b > ip6-allrouters: [icmp6 sum ok] ICMP
6, router solicitation, length 16
          source link-address option (1), length 8 (1): e4:5f:01:20:24:0b
            0x0000:  e45f 0120 240b
```

The router replies with the network details.

```
12:55:27.834684 60:8d:26:a7:c1:88 (oui Unknown) > 33:33:00:00:00:01 (oui Unknow
n), ethertype IPv6 (0x86dd), length 174: (hlim 255, next-header ICMPv6 (58) pay
load length: 120) bthub.home > ip6-allnodes: [icmp6 sum ok] ICMP6, router adver
tisement, length 120
        hop limit 64, Flags [other stateful], pref medium, router lifetime 180
s, reachable time 0ms, retrans timer 0ms
          prefix info option (3), length 32 (4): 2a00:23c5:ee00:5001::/64, Flag
s [onlink, auto, router], valid time 300s, pref. time 120s
            0x0000:  40e0 0000 012c 0000 0078 0000 0000 2a00
            0x0010:  23c5 ee00 5001 0000 0000 0000 0000
          prefix info option (3), length 32 (4): fd4d:869:6f93::/64, Flags [onl
ink, auto, router], valid time 10080s, pref. time 2880s
            0x0000:  40e0 0000 2760 0000 0b40 0000 0000 fd4d
            0x0010:  0869 6f93 0000 0000 0000 0000 0000
          rdnss option (25), length 24 (3):  lifetime 60s, addr: bthub.home
            0x0000:  0000 0000 003c fe80 0000 0000 0000 628d
            0x0010:  26ff fea7 c188
          mtu option (5), length 8 (1):  1492
            0x0000:  0000 0000 05d4
          source link-address option (1), length 8 (1): 60:8d:26:a7:c1:88
            0x0000:  608d 26a7 c188
```

The device sends an information request to the DHCP multicast address asking for the TFTP details.

```
12:55:27.838300 e4:5f:01:20:24:0b (oui Unknown) > 33:33:00:01:00:02 (oui Unknow
n), ethertype IPv6 (0x86dd), length 98: (hlim 255, next-header UDP (17) payload
length: 44) fe80::e65f:1ff:fe20:240b.dhcpv6-client > ff02::1:2.dhcpv6-server:
[udp sum ok] dhcp6 inf-req (xid=e5e0a4 (client-ID hwaddr type 1 e45f0120240b)
(option-request opt_59) (opt_61) (elapsed-time 0))
```

The DHCP server replies with the TFTP server details (`opt_59`).

```
12:55:27.838898 dc:a6:32:6f:73:f4 (oui Unknown) > e4:5f:01:20:24:0b (oui Unknow
n), ethertype IPv6 (0x86dd), length 150: (flowlabel 0xd1248, hlim 64, next-head
er UDP (17) payload length: 96) fe80::537a:52c:c647:b184.dhcpv6-server > fe80::
e65f:1ff:fe20:240b.dhcpv6-client: [bad udp cksum 0xd870 -> 0x78bb!] dhcp6 reply
(xid=e5e0a4 (client-ID hwaddr type 1 e45f0120240b) (server-ID hwaddr/time type
1 time 671211709 dca6326f73f4) (opt_59))
```

The device asks for the TFTP server MAC address since it can tell it's on the same network.

```
12:55:28.834796 e4:5f:01:20:24:0b (oui Unknown) > 33:33:ff:1d:fe:2a (oui Unknow
n), ethertype IPv6 (0x86dd), length 86: (hlim 255, next-header ICMPv6 (58) payl
```

```
oad length: 32) fe80::e65f:1ff:fe20:240b > ff02::1:ff1d:fe2a: [icmp6 sum ok] IC
MP6, neighbor solicitation, length 32, who has 2a00:23c5:ee00:5001:57f1:7523:2f
1d:fe2a
          source link-address option (1), length 8 (1): e4:5f:01:20:24:0b
            0x0000:  e45f 0120 240b
```

The FTP server replies with its MAC address.

```
12:55:28.834875 dc:a6:32:6f:73:f4 (oui Unknown) > e4:5f:01:20:24:0b (oui Unknow
n), ethertype IPv6 (0x86dd), length 86: (hlim 255, next-header ICMPv6 (58) payl
oad length: 32) 2a00:23c5:ee00:5001:57f1:7523:2f1d:fe2a > fe80::e65f:1ff:fe20:2
40b: [icmp6 sum ok] ICMP6, neighbor advertisement, length 32, tgt is 2a00:23c5:
ee00:5001:57f1:7523:2f1d:fe2a, Flags [solicited, override]
          destination link-address option (2), length 8 (1): dc:a6:32:6f:73:f4
            0x0000:  dca6 326f 73f4
```

The device starts making TFTP requests.

```
12:55:28.861097 e4:5f:01:20:24:0b (oui Unknown) > dc:a6:32:6f:73:f4 (oui Unknow
n), ethertype IPv6 (0x86dd), length 111: (hlim 255, next-header UDP (17) payloa
d length: 57) 2a00:23c5:ee00:5001:e65f:1ff:fe20:240b.46930 > 2a00:23c5:ee00:500
1:57f1:7523:2f1d:fe2a.tftp: [udp sum ok]  49 RRQ "ab5a4158/start4.elf" octet ts
ize 0 blksize 1024
```