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 may 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

It is possible to find the IP address of your Raspberry Pi without connecting to a screen using one of the following methods:

NOTE If you are using a display with your Raspberry Pi and if you boot to the command line instead of the desktop, your IP address should be shown in the last few messages before the login prompt. Otherwise open a Terminal window and type `hostname -I` which will reveal your Raspberry Pi's IP address.

Router devices list

In a web browser navigate to your router's IP address e.g. `http://192.168.1.1`, which is usually 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.

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 will also change 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.

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 be at addresses like `192.168.1.2`, `192.168.1.3`, `192.168.1.4`, 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 IPv6 addresses by pinging from a second device

First find your own IP address(es), in other words the one of the computer you're using to find your Raspberry Pi's IP address by `hostname -I`

`fd00::ba27:ebff:feb6:f293`
`2001:db8:494:9d01:ba27:ebff:feb6:f293`

The example shows two IP addresses. The first one is a so called unique local unicast address(`fc00::/7`). The second one is the global unicast address(`2000::/3`). It is also possible to see only one of them depending on your network (router) configuration. Both addresses are valid for reaching the Raspberry Pi within your LAN. The address out of `2000::/3` is accessible world wide, provided your router's firewall is opened.

Now use one of IPs from the first step to ping all local nodes:

```
ping -c 2 -I 2001:db8:494:9d01:ba27:ebff:feb6:f293  ff02::1
ping -c 2 -I 2001:db8:494:9d01:ba27:ebff:feb6:f293  ff02::1%eth0
```

`-c 2` stands for sending two echo requests

`-I` with the IP address, it sets the interface and the source address of the echo request, it is necessary to choose the interface's IP address, `eth0` isn't sufficient - the answer would be the local link address(`fe80::/10`), we need the global or local unicast address

`ff02::1` is a well known multicast address for all nodes on the link, so it behaves like a local broadcast, usually it is defined in `/etc/hosts` so you can also use the name (`ip6-allnodes` or `ipv6-allnodes`) instead of the literal address

Some newer systems expect the interface ID behind the multicast address.

```
ping -c 2 -I 2001:db8:494:9d01:ba27:ebff:feb6:f293 ip6-allnodes
PING ip6-allnodes(ip6-allnodes (ff02::1)) from
2001:db8:494:9d01:ba27:ebff:feb6:f293 : 56 data bytes
64 bytes from 2001:db8:494:9d01:ba27:ebff:feb6:f293: icmp_seq=1 ttl=64
time=0.597 ms
64 bytes from witz.fritz.box (2001:db8:494:9d01:728b:cdff:fe7d:a2e):
icmp_seq=1 ttl=255 time=1.05 ms (DUP!)
64 bytes from raspberrypi4.fritz.box (2001:db8:494:9d01:dea6:32ff:fe23:6be1):
icmp_seq=1 ttl=64 time=1.05 ms (DUP!)
64 bytes from 2001:db8:494:9d01:da37:beff:fefd:f09d
(2001:db8:494:9d01:da37:beff:fefd:f09d): icmp_seq=1 ttl=255 time=1.05 ms
(DUP!)
64 bytes from fusion.fritz.box (2001:db8:494:9d01:1e6f:65ff:fec9:8746):
icmp_seq=1 ttl=255 time=2.12 ms (DUP!)
64 bytes from fritz.box (2001:db8:494:9d01:464e:6dff:fe72:8a08): icmp_seq=1
ttl=64 time=2.62 ms (DUP!)
64 bytes from raspberrypi.fritz.box (2001:db8:494:9d01:ba27:ebff:feb6:f293):
icmp_seq=2 ttl=64 time=0.480 ms

--- ip6-allnodes ping statistics ---
2 packets transmitted, 2 received, +5 duplicates, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.480/1.283/2.623/0.735 ms
```

This should result in replies from all the nodes on your (W)LAN link, with associated DNS names.

Exclude your own IP(here `2001:db8:494:9d01:ba27:ebff:feb6:f293`), then check the others by trying to connect them via SSH.

```
ssh pi@2001:db8:494:9d01:dea6:32ff:fe23:6be1
The authenticity of host '2001:db8:494:9d01:dea6:32ff:fe23:6be1
(2001:db8:494:9d01:dea6:32ff:fe23:6be1)' can't be established.
ECDSA key fingerprint is SHA256:DAW68oen42TdWDyrOycDZ1+y5ZV5D81kaVoi5FnpvoM.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '2001:db8:494:9d01:dea6:32ff:fe23:6be1' (ECDSA) to
the list of known hosts.
pi@2001:db8:494:9d01:dea6:32ff:fe23:6be1's password:
Linux raspberrypi4 4.19.75-v7l+ #1270 SMP Tue Sep 24 18:51:41 BST 2019 armv7l

...

pi@raspberrypi4:~ $
```

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 bo partition of the SD Card. When the Raspberry Pi boots, it looks for the `ssh` file. If it is found, SSH is enabl 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 tha 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,

```
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.

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 pi@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:

```
tail -f /var/log/secure
# might return:
Nov 23 12:31:26 raspberrypi sshd[9146]: Authentication refused: bad ownership
or 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 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_check)
/usr/local 192.168.0.10(rw,sync,no_subtree_check)
192.168.0.11(rw,sync,no_subtree_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

By default, `rpcbind` only binds to the loopback interface. To enable access to `rpcbind` from remote machines, you need to change `/etc/conf.d/rpcbind` to get rid of either `-l` or `-i 127.0.0.1`.

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
```