Part 40
-
Network Booting
Of Your Raspberry Pi

Version: 2019-12-08

Upfront security information

The default user name for Raspbian is **pi** with its password **raspberry**
The root password is disabled.

When using SSH (port 22), select an Xterm terminal in color, set white on black as screen colors, use UTF-8 as character set.

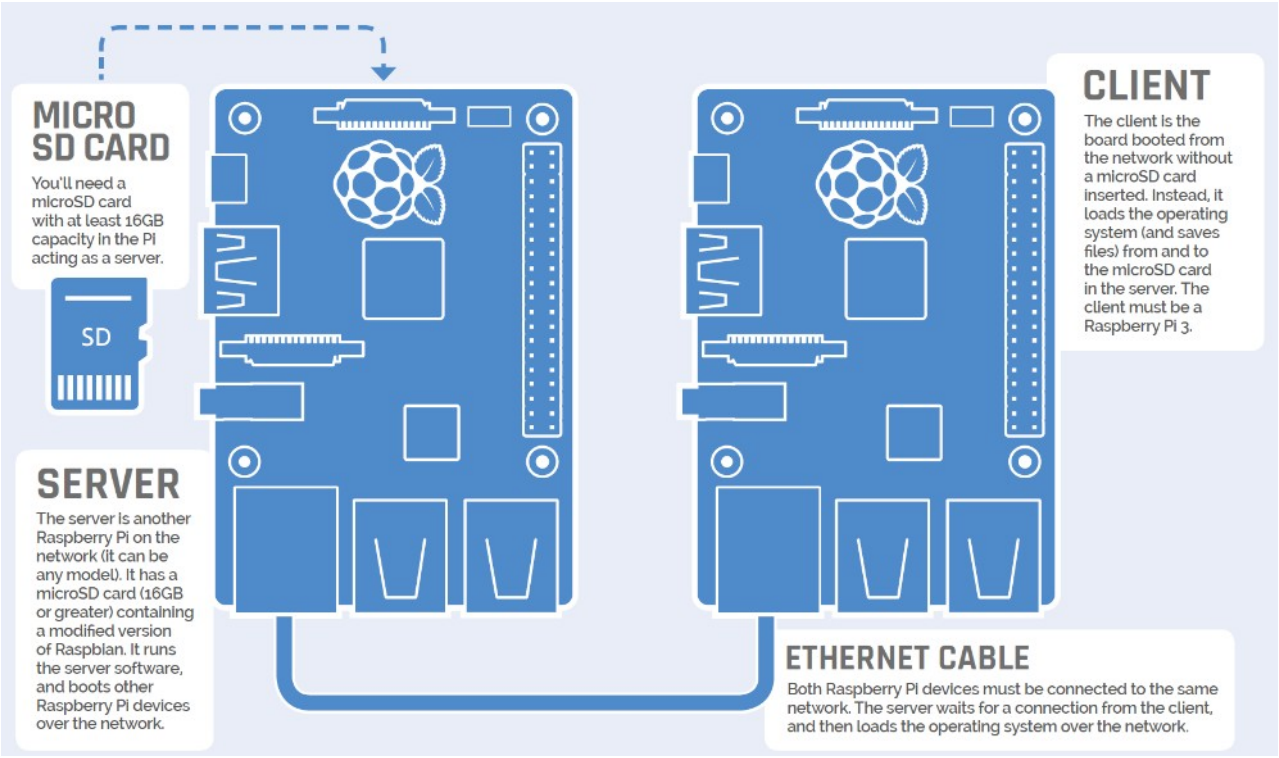
Network boot your Raspberry Pi

With this feature enabled, you can install Raspberry Pi on one computer and then use it to boot another Raspberry Pi's on the same network.
In particular and minimal, you need two boards. One acts as the server (booted using a microSD card containing Raspbian); the other is the client which will boot into Raspbian without any attached storage device.
Only one of the Raspberry Pi devices (the server) requires a microSD card, but it has to be at least 16GB capacity. This is because it must carry the image of Raspbian used to boot the board, and another image of Raspbian used to boot the computers on the network.
"The big advantage with network booting," says Gordon Hollingworth, Raspberry Pi's director of engineering, "is, with a small penalty for booting time, you can boot a whole bunch of Raspberry Pi boards from a single server, with absolutely no programming of SD cards."
"We use the network booting with our test subsystem," explains Gordon. "This allows us to completely reboot a Pi with no reliance on old boot software". So if they break something during testing, they can just reload the operating system.
"Since you can easily create a shared filing system," he continues, "it makes it very easy to add Raspberry Pis to a server to provide a whole network of Raspberry Pis, with no fiddly SD cards."

This tutorial explains how to set up a simple DHCP/TFTP server which will allow you to boot a Raspberry Pi 3 from the network.
This assumes you have an existing home network and want to use a Raspberry Pi for the server.

You'll need:

- Two Raspberry Pi boards (at least one needs to be a Raspberry Pi 3)
- Ethernet cable
- 16GB (or larger) microSD card



Client one-time setup:
Install Raspbian Lite (or the full OS if you want) from the Downloads page onto an SD card, using Win32DiskImager if you're on Windows, or dd if you're on Linux/Mac.
Place the microSD card into the client Raspberry Pi (the Raspberry Pi 3 that you intend to boot over the network and without the card in future). Just as with USB boot, you'll first need to prepare the `/boot` directory with experimental boot files. If you're using Raspbian Lite, you need to run a `rpi-update` before you can use it:

```
sudo apt-get update
sudo apt-get install BRrpi-update
sudo BRANCH=next rpi-update
```

Then enable USB boot mode with:

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

This adds 'program_usb_boot_mode=1' to the end of `/boot/config.txt`. Reboot the Raspberry Pi:

```
sudo reboot
```

Once the client Pi has rebooted, check that the OTP (one-time programmable) memory has been programmed using:

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

Ensure the output is `'3020000a'`. The client configuration is almost done. The final thing to do is to remove the 'program_usb_boot_mode' line from config.txt:

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

Scroll down to the end and remove the line marked `'program_usb_boot_mode=1'`. Press Ctrl+O and hit Enter to save the file, then Ctrl+X to return to the command line.
Finally, shut down the client Pi with

```
sudo poweroff.
```

Server setup:
Remove the microSD card and transfer it to the server Raspberry Pi. Power it up and immediately expand the root file system, using `raspi-config`, to take up the entire SD card and reboot.
The client Raspberry Pi needs a root file system to boot from, which has to be separate from the file system being used by the server. So before we do anything else on the server, we're going to make a full copy of its file system and put it into a directory called `/nfs/client1`.

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

Regenerate the SSH host keys on the client file system 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
```

We need to logout and login again. Do this using

```
exit
```

Now unmount the volumes

```
sudo umount dev
sudo umount sys
sudo umount proc
```

Now you need to find the settings of your local network. First, locate the address of your router (or gateway), which you can find with:

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

This will be a four-digit number, and typically it ends in 1. Ours is `192.168.0.1`.
Next, we need to find the IP address of our server Raspberry Pi on the network. This will be the same address, but with a different last number.

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

This should give an output like:

```
inet 192.168.0.197/24 brd 192.168.0.255 scope global eth0
```

The first address (inet) is the IP address of your server Pi on the network. Ours is device 197.
The part after the slash is the network size. It's highly likely that yours will be a /24.
The second part (the brd) is the network size (the number of total devices allowed on the network, which is almost always 255).
Write down both the inet and brd numbers.
Finally, note down the address of your DNS server. You can find this with:

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

You will see 'nameserver' followed by another number. You may have more than one nameserver address. We get:

```
nameserver 194.168.4.100
nameserver 194.168.8.100
```

Write down the nameserver address(es).

Configuring the server:
Now we're going to configure a static network address on your server Raspberry Pi using

```
sudo nano /etc/network/interfaces
```

Change the line '`iface eth0 inet manual`' so that the address is the inet you wrote down, the netmask address is `255.255.255.0`, and the gateway address is the number received using ip route. Ours looks like this:
auto eth0

```
iface eth0 inet static
    address 192.168.0.197
    netmask 255.255.255.0
    gateway 192.168.0.1
```

Use Ctrl+O, Enter, and Ctrl+X to save the text file and return to the command line.
Next, we disable the DHCP client daemon and switch to standard Debian networking:

```
sudo systemctl disable dhcpcd
sudo systemctl enable networking
```

Reboot for the changes to take effect:

```
sudo reboot
```

At this point, you won't have working DNS, so you'll need to add the server you noted down before, to `/etc/resolv.conf`. Do this by using the following command, where the IP address is that of the router/gateway address you found earlier:

```
echo "nameserver 192.168.0.1" | sudo tee –a /etc/resolv.conf
```

Then make the file immutable (because otherwise `dnsmasq` will interfere) with the following command:

```
sudo chattr +i /etc/resolv.conf
```

Next, we're going to install some software we need:

```
sudo apt-get update
sudo apt-get install dnsmasq tcpdump
```

Stop `dnsmasq` breaking DNS resolving:

```
sudo rm /etc/resolvconf/update.d/dnsmasq
sudo reboot
```

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

```
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, '`BOOTP/DHCP, Request from ...`'. It will have lines that look like:

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

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

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

Replace the contents of `dnsmasq.conf` with:

```
port=0
dhcp-range=192.168.0.255,proxy
log-dhcp
enable-tftp
tftp-root=/tftpboot
pxe-service=0,"Raspberry Pi Boot"
```

Make sure the first address of the dhcp-range line is the broadcast address (brd) number you noted down earlier (the number typically ending in 255). 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:

```
tail -f /var/log/daemon.log
```

You should see several lines like this:

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

Next, you'll need to copy `bootcode.bin` and `start.elf` into the `/tftpboot` directory; you should be able to do this by just copying the files from /boot, since they are the right ones. We need a kernel, so we might as well copy the entire boot directory.
First, use Ctrl+Z to exit the monitoring state. Then type the following:

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

Restart `dnsmasq` for good measure:

```
sudo systemctl restart dnsmasq
```

Set up root:
We now have a Raspberry Pi that boots until it tries to load a root file system (which it doesn't have). All we have to do to get this working is to export the `/nfs/client1` file system we created earlier:

```
sudo apt-get install nfs-kernel-server
echo "/nfs/client1 *(rw,sync,no_subtree_check,no_root_squash)" | sudo tee -
a /etc/exports
sudo systemctl enable rpcbind
sudo systemctl restart rpcbind
sudo systemctl enable nfs-kernel-server
sudo systemctl restart nfs-kernel-server
```

We need to edit the `cmdline.txt` file:

```
sudo nano /tftpboot/cmdline.txt
```

From 'root=' onwards, replace it with:

```
root=/dev/nfs nfsroot=192.168.0.197:/nfs/client1 rw ip=dhcp rootwait
elevator=deadline
```

You should substitute the IP address here with the inet address you noted down earlier.
Finally, enter:

```
sudo nano /nfs/client1/etc/fstab
```

Remove the '`/dev/mmcblkp1`' and '`p2`' lines (only 'proc' should be left).

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