

Getting Started

Raspberry Pi Operating System

As stated on the "Home" page of this guide, it is assumed that you are running a Raspberry Pi as your main computer for your printer. If you are not then this information may be worth reading, but also may not be directly relevant.

There is a known issue with Raspberry Pi operating systems where "Bullseye" based distros (version 6.1.something) and 64bit can cause very strange timing issues which can then lead to Timer Too Close, Missed Scheduling, or other errors. A simple way to check what version operating system you have is to SSH in and then run:

```
uname -a
```

This will give you information about your OS. You are looking for the version number and the architecture.

```
Linux trident 6.1.21-v8+ #1642 SMP PREEMPT Mon Apr 3 17:24:16 BST 2023 aarch64
```

If you have 6.1.something as the version and aarch64 as the architecture, then it would be best to backup any existing Klipper data (config files and maybe moonraker history if you want to keep that) and then reflashing the SD card with the latest Raspberry Pi OS. Eric Zimmerman has fantastic instructions for doing both of those steps [here](#).

If you find your version is 6.6.something, or 5.something, then it is fine. Also, if your architecture is `armv7l` then it means you are running a 32bit OS and this is also unaffected.

Network Service, CAN Speeds, and Transmit Queue Length

In order to dictate the speed at which your CAN network runs at you will need make sure there is a service on the Pi that can configure a CANBus interface, and also the necessary information in configuration files so this service knows what settings you want to use.

To set everything up, SSH into your pi and run the commands needed for your network setup:

Wait, wasn't this section different before?

Yes, originally the "default" method of configuring the CAN network was using a can0 file and the ifupdown network service. However, ifupdown is being phased out by newer linux distributions. The more universal option seems to be using the systemd-networkd service instead. The good news is that this seems to be installed (if not enabled) on most linux distros we'd see on 3d printers. So streamlining the process down should make it better for everyone.

Make sure the systemd-networkd service is enabled by running

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

then start the service with:

```
sudo systemctl start systemd-networkd
```

If you get a `Failed to start systemd-networkd.service: Unit systemd-networkd.service is masked` error (or similar), run `sudo systemctl unmask systemd-networkd` first to unmask it, then run `sudo systemctl start systemd-networkd` again

then check it is running properly with:

```
systemctl | grep systemd-networkd
```

and make sure it shows as "loaded active running"

```
pi@raspberrypi:~ $ systemctl | grep systemd-networkd
systemd-networkd.service
loaded active          running               Network Configuration
```

Unfortunately, this also enables the systemd-networkd-wait-online service which can cause your system to be "frozen" for around 2 minutes on boot as it waits for network interfaces to be online. If

the problem was only the CAN interface there would be ways around this but as there are edge cases where the wait would still happen I've found it's best to just disable the wait-online service entirely by running:

```
sudo systemctl disable systemd-networkd-wait-online.service
```

If disabling this service results in any negative impact on certain linux distros or setups please let me know via an [issue ticket](#).

Then configure the txqueuelen for the interface by running the following command (copy the line entirely and paste it into your SSH session)

```
echo -e 'SUBSYSTEM=="net", ACTION=="change|add", KERNEL=="can*" ATTR{tx_queue_len}="128"' | sudo tee /etc
```

To confirm it has applied correctly, run

```
cat /etc/udev/rules.d/10-can.rules
```

and it should look like this:

```
pi@raspberrypi:~$ echo -e 'SUBSYSTEM=="net", ACTION=="change|add", KERNEL=="can*" ATTR{tx_queue_len}="128"' | sudo tee /etc/udev/rules.d/10-can.rules > /dev/null
pi@raspberrypi:~$ cat /etc/udev/rules.d/10-can.rules
SUBSYSTEM=="net", ACTION=="change|add", KERNEL=="can*" ATTR{tx_queue_len}="128"
```

Didn't this use to be a txqueuelen of 1024?

Yes, for a long time it was recommended to use 1024 for the transmit queue length, and if that is working for you then power on. But it is recommended directly from Kevin O'Connor (of Klipper fame) to use a lower queue length of 128 and in my testing I've found no issues doing so. Also it should *theoretically* help with some timeout issues using a smaller queue though I haven't seen real evidence one way or another. Still, seems good to align with the recommended Klipper docs on the matter.

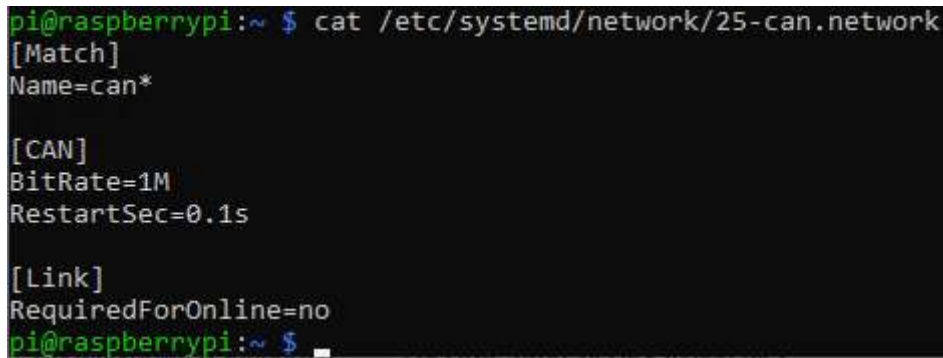
Now finally, to enable the can0 interface and set the speed run the following command:

```
echo -e "[Match]\nName=can*\n\n[CAN]\nBitRate=1M\nRestartSec=0.1s\n\n[Link]\nRequiredForOnline=no" | sudo
```

To confirm it has applied correctly, run

```
cat /etc/systemd/network/25-can.network
```

and it should look like this:

A terminal window on a Raspberry Pi showing the contents of the file /etc/systemd/network/25-can.network. The output is as follows:

```
pi@raspberrypi:~ $ cat /etc/systemd/network/25-can.network
[Match]
Name=can*

[CAN]
BitRate=1M
RestartSec=0.1s

[Link]
RequiredForOnline=no
pi@raspberrypi:~ $
```

Then finally reboot the Pi (so the systemd-networkd service has started properly reading from the config files) with a:

```
sudo reboot now
```

120R Termination Resistors

A CANBus expects both ends of the CanH/CanL wire run to be terminated with a 120 ohm resistor bridging the High and Low wires. Your CAN board will almost certainly have provisions for this somewhere.

You want to have **two** of these termination resistors in your CANBus circuit. **No more, No less.** Running with too many connected can be just as bad as running with none.

Put these at the physical “end” of the CAN wires.

If you want a more in depth breakdown of the termination resistors, or you have a CAN setup that seems more complicated than what is outlined above, have a read of the [termination resistors](#) page.

CAN Adapter/Mainboard

Some boards (Like the BTT Octopus) have the 120 ohm resistor permanently connected across the CanH/L wires, so nothing you need to do there. Others will have a two-pin header (sometimes labelled “120R”) that you can put a jumper on and this will bring the termination resistor into the circuit.

The same can be said for dedicated USB CAN adapters (like the U2C). Most will have a header that you can put a jumper on to enable the resistor.

You can find information and diagrams on the 120 ohm termination resistor placement for boards in the *mainboard common hardware* section.

Toolhead

Nearly all Toolheads will have a two-pin header (sometimes labelled 120R) that you can put a jumper on to bring the 120 ohm resistor into the circuit.

You can find information and diagrams on the 120 ohm termination resistor placement for boards in the *toolhead common hardware* section.

Cabling

There is a high likelihood that with your changeover to canbus you will also change to some sort of umbilical system for your toolhead. This isn't absolutely necessary, but extremely common. Some boards even come with a premade cable which may or may not work well in cable chains anyway.

Before getting too swept up in what the absolute best gold plated gucci cabling would be for your system, keep the following things in mind. They are the most important factors when it comes to canbus toolhead board cabling. Nearly every single time someone has issues (after the initial install) it is because a wire has cracked or connector crimp is failing or similar, so take extra careful to follow these steps:

1 Good Crimps

If you are making your own cable then be extra sure your wire crimps are properly done. A loose crimp can fall off, a bent crimp can crack. And make sure the pins are fully seated into the connector housing. Do a pull test to be sure, it's very easy for a pin to look inserted but not actually clicked in and they can get pushed out the back during use.

This also applies to premade cables. I've seen some where the H/L connectors get pushed back slightly in the housing, enough that it causes intermittent connection issues.

2 Strain Relief

No matter what you do **not** want the cable having any sort of movement at/near the connector to the toolhead board. If the wires at those crimp points start moving then it's only a matter of

time before they'll crack.

Have the cable run away from the toolhead for a short distance and then secure it down so no matter what the rest of the umbilical is doing there is no movement transmitted directly to the connector.

3 Flexible Cable

I know I said not to get worked up about über cables, and that's still true, but also don't go using old solid core mains wiring you ripped from the walls. You want the cable/wires to be flexible enough to withstand the constant movement of a printer. That means don't go *too* thick on the wire gauge, and also make sure it is decently stranded wire (not 7-strand high current wire).

22AWG or thicker (smaller number is thicker wire) is usually fine for the power wires. The CAN H/L can be thinner, 24 or 26AWG is common, but you can also just do four 22AWG wires or something without trouble as well.

A common method of supporting and strain relieving the umbilical is to use PG7 glands, and although these work really well they also tend to mean people have to cut or repin the premade cables that come with some boards. Because of this I prefer using the printable [P.U.G](#) umbilical gland. It does the same job as a PG7 but comes in two halves so you can clamp it over any cable without needing to cut/depin anything.

I've got a public Printables "collection" of various PUG mounts for different extruders [here](#) and will hopefully grow it over time, so be sure to check there to see if one of them fits your toolboard/extruder.

Next Step

Now that the can0 interface files are set up, you need to choose how you are going to connect your Pi to the CANBus network.

To actually create a CAN network in your system, your Pi needs some sort of device to act as a CAN adapter (think of it like a USB network card, or USB wifi dongle).

The simplest plug-and-play option is to use a dedicated USB to Can device such as the BigTreeTech U2C, Mellow Fly UTOC, Fysetc UCAN, etc. (other devices exist as well). [Click here if you have a Dedicated USB-CAN Adapter](#) to continue the guide.

The second “cheaper” option is to actually utilise your printer mainboard (ie. Octopus, Manta, Spider, etc.) to function as a usb-can bridge for klipper. [Click here if you have are using your mainboard as the CAN interface](#) to continue the guide.