https://elinux.org/RPi_Serial_Connection

RPi Serial Connection

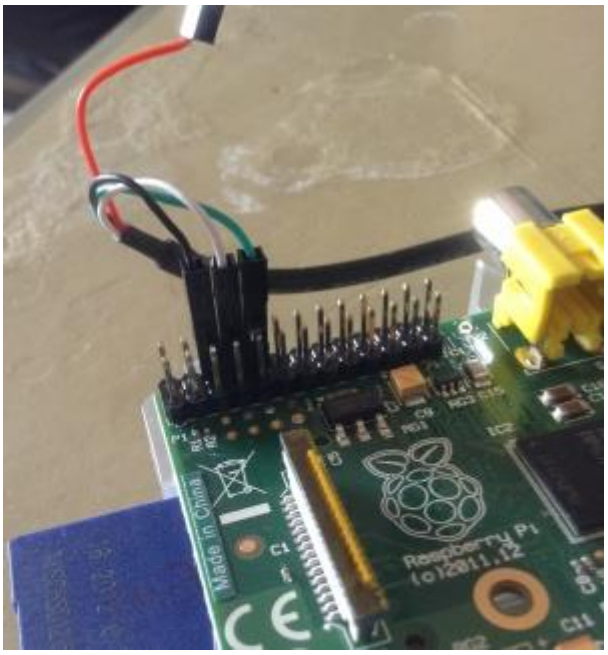
Back to RPi Advanced Setup.

The serial port is a low-level way to send data between the Raspberry Pi and another computer system. There are two main ways in which it can be used:

- Connecting to a PC to allow access to the Linux console. This can help to fix problems during boot, or to log in to the Raspberry Pi if the video and network are not available.
- Connecting to a microcontroller or other peripheral which has a serial interface. This can be useful if you want the Raspberry Pi to control another device.

Contents

Connections and signal levels



Adafruit serial cable connected to Pi

The Raspberry Pi serial port consists of two signals (a 'transmit' signal, TxD and a 'receive' signal RxD) made available on the GPIO header. To connect to another serial device, you connect the 'transmit' of one to the 'receive' of the other, and vice versa. You will also need to connect the Ground pins of the two devices together.

The Broadcom chip at the heart of the Raspberry Pi uses 0 and 3.3 V logic levels, not the +/- 12 V used by RS-232 serial ports found on some older PCs. If you wish to connect one of these, you need a board or adapter to convert the signal levels. See this tutorial for one example on how to build a 3.3 V to RS-232 level converter with a breadboard, a MAX3232CPE IC and five 0.1 µF capacitors.

If you wish to connect your Raspberry Pi to a PC with a USB port, the simplest option is to use a USB-to-serial cable which uses 3.3 V logic levels (e.g. the Adafruit 954 cable, the FTDI TTL-232R-RPI cable, or the Debug Buddy ultimate serial port). These can be simply plugged in directly to the GPIO header (see illustration).

- For using the *Adafruit 954* cable on Windows, see Adafruit 954 USB serial cable.

If you wish to connect to a peripheral which has 0/5 V signals, you should ideally have a circuit to convert between the voltage levels. See this tutorial for an example using a ready-made level shifter module. Other circuits for level shifting are shown at RPi_GPIO_Interface_Circuits#Level_Shifters. The Debug Buddy ultimate serial port can also be configured for 0/5 V signals.

NOTE FOR RASPBERRY PI 3: The Raspberry pi 3 has changed things a bit and you might need to add the option `enable_uart=1` at the end of `/boot/config.txt` (see this post by a Pi Engineer)

Connection to a PC

You can connect the Raspberry Pi to a PC using a USB-serial cable, or (if it has an RS-232 port) a level-converter circuit - see above for details. When this is done, you will need to set up a terminal emulator program on your PC as described below.

Console serial parameters

The following parameters are needed to connect to the Raspberry Pi console, and apply on both Linux and Windows.

- Speed (baud rate): 115200
- Bits: 8
- Parity: None
- Stop Bits: 1
- Flow Control: None

Linux terminal set up

If your PC is running Linux, you will need to know the *port name* of its serial port:

- Built-in (standard) Serial Port: the Linux standard is **/dev/ttyS0**, **/dev/ttyS1**, and so on
- USB Serial Port Adapter: **/dev/ttyUSB0**, **/dev/ttyUSB1**, and so on.
    - Some types of USB serial adapter may appear as **/dev/ttyACM0** ...

You will need to be a member of the *dialout* group to access this port (for later releases the required group is *tty*). You can check which is needed with:

```
ls -l /dev/ttyUSB0
```

and you will see something like "crw-rw----T 1 root dialout ...", *c* means character device, and root can 'read,write' and the group *dialout* can 'read,write' to the port and everyone else cannot access it.

To find out if you, the current user, is in the group dialout, use the command:

```
id
```

If you do not see *dialout* listed, add yourself with the command

```
sudo usermod -a -G dialout username
```

You then have a choice of terminal emulation programs:

- **Super Easy Way Using GNU Screen**

Enter the command below into a terminal window

```
 screen port_name 115200
```

To exit GNU screen, type Control-A k.

- **Super Easy Way Using Minicom**

Run Minicom with the following parameters:

```
minicom -b 115200 -o -D Port_Name
```

You can exit Minicom with Control-A x

Note: If you haven't configured minicom before (i.e: first use after installation), or if you find that your keyboard key presses are not sent to the RPi, you should make sure **Hardware Flow Control** is disabled. See **Tedious Old-Fashioned Way Using Minicom** to configure minicom.

- **Tedious Old-Fashioned Way Using Minicom**

Another method to setup *minicom* is described in the Tincantools Minicom Tutorial

- **GUI method with GtkTerm**

Start *GtkTerm*, select Configuration->Port and enter the values above in the labeled fields.

Network connection with the point-to-point protocol (ppp)

The easiest way to set up a network connection between your Raspberry Pi and another computer is with an Ethernet cable. If this is not possible, as is the case for the Raspberry Pi Model A, you can set up a connection over the serial cable. This uses the Point-to-point Protocol (PPP). A network connection running over a serial cable can be very useful for copying files onto the Raspberry Pi.

Step 1: Log in to the Raspberry Pi over the serial cable and run the Point-to-Point Protocol Daemon:

```
 sudo pppd noauth
```

Some garbage will start appearing in the terminal. This is the cue to quit your terminal program and proceed to step two.

Step 2: On your local computer, start the Point-to-Point protocol. On a Linux or Mac computer you can do this by typing:

```
 sudo pppd noauth proxyarp /dev/tty.usbserial-FTGCC2MV 115200
10.0.0.1:10.0.0.2 passive local maxfail 0 nocrtscts xonxoff
```

replacing /dev/tty.usbserial-FTGCC2MV with the name of your serial port. In the above line, 115200 is the baud rate of the connection, 10.0.0.1 is the local internet protocol (IP) address, the address you want your computer to have. 10.0.0.2 is the remote IP address, it is the address that the Raspberry Pi will have.

Test the connection:

```
 ping 10.0.0.2
```

Virtual connection to the LAN

Instead of 10.0.0.0/8 you could as well use normal 192.168.0.0/16 addresses; the first address must be the real address of the local (serving) system. You can chose the second address; it must not yet be assigned on the LAN (and be outside the DHCP range). The advantage is that the system connected to the serial line will appear as if it is directly connected to the LAN (arp protocol).

You must enable routing on the system directly connected to the LAN for other systems to access the system connected to the serial line:

```
sudo sysctl -w net.ipv4.ip_forward=1
```

On the guest system connected via the serial cable you must set the default route pointing to the serving system, e.g.

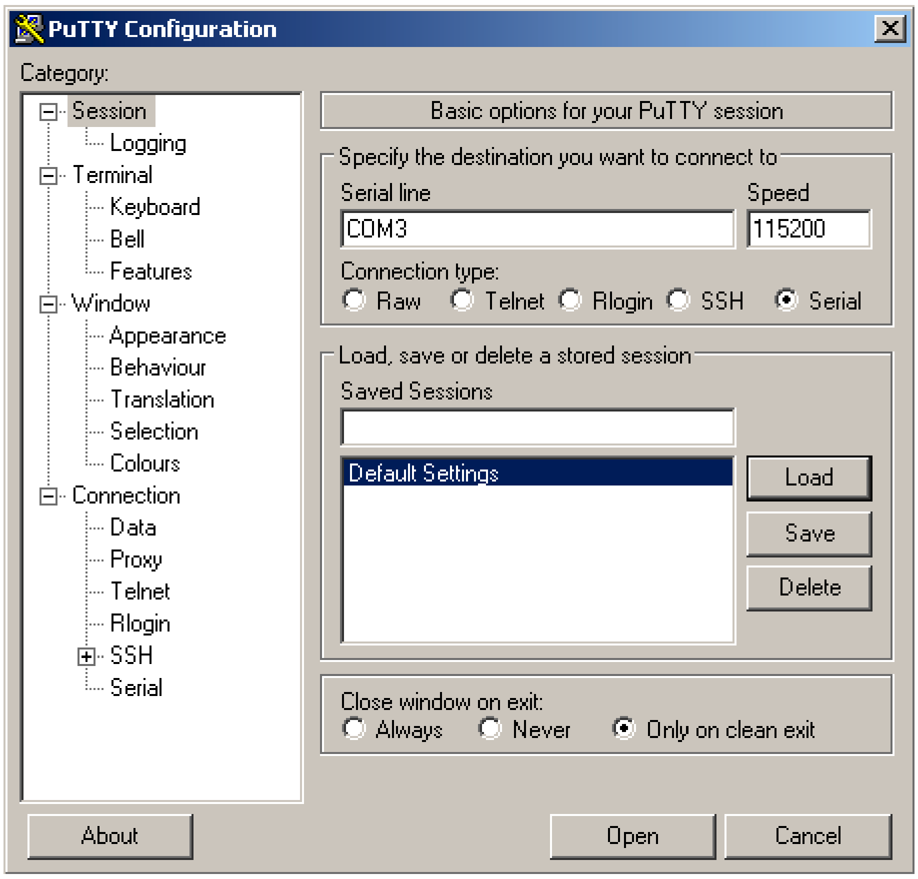
```
sudo route add default gateway 192.168.1.21
```

You should also configure /etc/resolv.conf if you want to use DNS.

Windows terminal set-up

Users of Windows Vista or later will need to download a terminal program, for instance PuTTY, or TeraTerm. Users of Windows XP and below can choose between using *PuTTY* and the built-in *Hyperterminal*.

PuTTY users simply need to choose 'serial', select the correct COM port and set the speed, as shown in the dialog below.



Specify the destination you want to connect to
Serial line: COM3
Speed: 115200
Connection type: Raw Telnet Rlogin SSH [✓ Serial]

Load, save or delete a stored session
Saved Sessions
Default Settings

Close window on exit: Always Never [✓ Only on clean exit]

If you are unsure of the COM port, run [Device Manager] and look under 'Ports'. USB-attached serial adapters should have the name of the adapter shown (the Adafruit cable comes up as 'Prolific USB-to_Serial Comm Port'.

Boot messages

If your connection is set up correctly, when the Raspberry Pi is booted you should see many messages as the system comes up:

```
Uncompressing Linux... done, booting the kernel.
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Linux version 3.2.27+ (dc4@dc4-arm-01) (gcc version 4.7.2
20120731 (prerelease) (crosstool-NG linaro-1.13.1+bzr2458 - Linaro GCC
2012.08) ) #250 PREEMPT Thu Oct 18 19:03:02 BST 2012
[    0.000000] CPU: ARMv6-compatible processor [410fb767] revision 7
(ARMv7), cr=00c5387d
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT nonaliasing
instruction cache
[    0.000000] Machine: BCM2708
[    0.000000] Memory policy: ECC disabled, Data cache writeback
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on.
Total pages: 113792
[    0.000000] Kernel command line: dma.dmachans=0x7f35
bcm2708_fb.fbwidth=656 bcm2708_fb.fbheight=416 bcm2708.boardrev=0xf
bcm2708.serial=0xcc5c4b6d smsc95xx.macaddr=B8:27:EB:5C:4B:6D sdhci-
bcm2708.emmc_clock_freq=100000000 vc_mem.mem_base=0x1c000000
vc_mem.mem_size=0x20000000  dwc_otg.lpm_enable=0 console=ttyAMA0,115200
console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline rootwait
```

and so on. Eventually, you should see a login prompt:

```
Debian GNU/Linux Wheezy/sid raspi2 ttyAMA0

raspi2 login:
```

You can then log in as you would with a keyboard and screen.

Unwanted serial garbage input

Note that on older software by accident the internal pullups of the RxD GPIO pins were not enabled, this could lead to lots of serial garbage being picked up if the GPIO pin was touched, or even if a finger was nearby. In extreme case this could lead to kernel warnings and other problems.

Connection to a microcontroller or other peripheral

H/W considerations

If your microcontroller or peripheral works with **5V logic levels**, level conversion is necessary - see 'Connecting to a PC' for details.

If your microcontroller or peripheral works with **3.3V logic levels** then you can connect its TxD, RxD signals directly to the RxD and TxD pins of the Raspberry. However it's probably a good idea to connect the signals with a 2.2 kΩ resistors in series. This will prevent damage if two outputs are accidentally connected together (e.g. if you connect TxD with TxD or if a GPIO input pin is accidentally programmed as output).

S/W: Preventing Linux from using the serial port

By default Linux will grab the serial port and use it as a terminal. If you want to use it for other purposes you must prevent this. Here are the methods you can use:

Method 1, raspi-config (easiest, try this first)

Run `sudo raspi-config` and check if it has the option `advanced options` -> `serial`. If it has, set it to disabled and you're done.

Method 2, using an existing script (easy)

There's a nice little script to automate the steps bellow.

Method 3, manual configuration (complex)

If neither raspi-config nor the script works for you then follow the hard way.

NOTE FOR RASPBERRY PI 3: The Raspberry pi 3 has changed things around a bit: ttyAMA0 now refers to the serial port that is connected to the bluetooth. The old serial port is now called ttyS0. So if you have an RPI3, everywhere you see "ttyAMA0" below, you should read "ttyS0".

The Broadcom UART appears as `/dev/ttyAMA0` under Linux. There are several minor things in the way if you want to have dedicated control of the serial port on a Raspberry Pi.

- Firstly, the kernel will use the port as controlled by kernel command line contained in `/boot/cmdline.txt`. The file will look something like this:

```
dwc_otg.lpm_enable=0 console=ttyAMA0,115200 kgdboc=ttyAMA0,115200
console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline rootwait
```

The *console* keyword outputs messages during boot, and the *kgdboc* keyword enables kernel debugging. You will need to remove all references to ttyAMA0. So, for the example above `/boot/cmdline.txt`, should contain:

```
dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4
elevator=deadline rootwait
```

You must be root to edit this (e.g. use `sudo nano /boot/cmdline.txt`). Be careful doing this, as a faulty command line can prevent the system booting.

- Secondly, after booting, a login prompt appears on the serial port. This is controlled by the following lines in `/etc/inittab`:

```
#Spawn a getty on Raspberry Pi serial line
```

```
T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

You will need to edit this file to comment out the second line, i.e.

```
#T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

Finally you will need to reboot the Raspberry Pi for the new settings to take effect. Once this is done, you can use `/dev/ttyAMA0` like any normal Linux serial port, and you won't get any unwanted traffic confusing the attached devices.

The above instructions have been verified on Raspbian 'Wheezy'; other distributions may be set up differently. To double-check, use

```
cat /proc/cmdline
```

to show the current kernel command line, and

```
ps aux | grep ttyAMA0
```

to search for `getty` processes using the serial port.

A tutorial on accessing the Raspberry Pi's serial port from Python is available at Serial_port_programming.

Handshaking lines

You can have the RTS0 signal on GPIO 17 (P1-11) or GPIO 31 (P5-06) if you set them to ALT function 3. Likewise, the CTS0 is available on GPIO 30 (P5-05), if it is set to ALT function 3. You can control the settings of I/O pins with gpio_setfunc.

Glitch when opening serial port

When the serial port is opened the voltage on TXD pulses negative for approximately 32 µs (regardless of the baud rate). This pulse may be interpreted as a transmission by a device connected to the TXD pin, which could have unintended effects. An error tolerant communication protocol should be used to avoid problems this glitch could cause. Another method for avoiding problems is to use a GPIO pin to implement the RTS signal, and to have the connected device ignore all data on TXD until RTS is asserted. If the connected device is susceptible to the glitch and cannot be modified, it is sometimes possible to obtain correct operation by opening the serial port in advance of initiating transmission. This can be done in the shell with the sleep program:

```
sleep infinity >/dev/ttyAMA0 &
```

In a shell script, the following commands may be used to kill the sleep process once serial transmission is complete.

```
sleep infinity >/dev/ttyAMA0 &
sleep_pid=$!
disown $sleep_pid # this is required to prevent an error message when sleep
is terminated
```

```
# Use serial port here
kill $sleep_pid
```