# *Preface*

Computer security is a difficult topic to learn. Unlike other areas of science and math, we know few fundamental principles that guarantee systems will remain secure. Although there is plenty to say about security, threats constantly emerge in new and surprising ways.

No vulnerability demonstrates this fact better than the *Rowhammer exploit*: with enough time to carry out an attack, this computer vulnerabilty effectively gives attackers full control of a computer system [**?**]. Was this exploit the product of lazy or stupid engineers? No. Attacks like Rowhammer exploit a fundamental tool we employ to reign in complexity: abstraction. Abstraction provides guarantees in order to simplify systems design. Unfortunately, many "guarantees" have surprising and unexpected corner cases. Case in point: Rowhammer exploits the physical structure of a computer's memory circuitry. This vulnerability exists in a domain that software engineers cannot be expected to know. In fact, no software technique can totally stop it.

Fortunately, all is not lost. In this course, you will develop a *defensive programming mindset*. When you program defensively, you assume that guarantees are imperfect. Most importantly, you assume that programmers—including you—make mistakes. Instead of making attacks *impossible*, which requires monumental efforts and is itself sometimes impossible, we focus on making attacks *expensive*. We want attackers to look at our systems and say "this is not worth the trouble." Making attacks inconvenient turns out not just to be easier to achieve—fully verifying that software is free of bugs remains an open research problem—experience suggests that it is the most effective defensive technique.

To program defensively, you must know the common pitfalls but be constantly on the lookout for new ones. We develop three key skills:

- knowing how to read and assess security literature,
- developing a toolbox of defensive programming techniques, and
- deploying this knowledge to prioritize effort on developing the most effective mitigations.

At the end of this class, you will not be a security expert. Nevertheless, I hope that I will have given you the key skills you need to *become* a security expert.

Happy hunting!

1

# Lab 0: Setting up your Raspberry Pi

*This course uses Raspberry Pi computers for all assignments. The rationale is to provide you with a machine for which you have total control, something that is difficult to do with shared lab computers. It also serves as a common platform to facilitate grading. Don't spend too much time worrying about this machine. Accidentally damaging it is inconvenient but not expensive. At the end of the semester, you can repurpose your Raspberry Pi for your own personal projects: it's yours!*

## 1.1 Learning Goals

In this lab, you will learn:

- The purpose of each component in your lab kit.
- How to use a serial console.
- How to install an operating system on your Raspberry Pi.

## 1.2 The Lab Kit

You were given a small lab kit at the start of this course. It contains a number of objects. What are these things?

*The Computer.* The most important item in your kit is the Raspberry Pi computer.

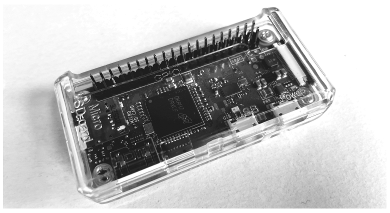


Figure 1.1: A computer.

| | |
|---|---|
| Model | Raspberry Pi Zero WH |
| Processor | ARMv6 at 1GHz |
| Memory | 512MB |
| Persistent storage | SD card slot |
| Video | 1080p mini-HDMI on Broadcom BCM2835 |
| Wifi | 2.4 GHz and 5 GHz 802.11b/g/n/ac wireless LAN |
| Expansion | 40-pin general-purpose input/output port (GPIO) |

Table 1.1: To put these specifications in context, your Raspberry Pi is about as fast as a good personal computer from around the time I graduated from college. While computers are certainly faster now, those computers were quite capable. Yes, I am old.

*The Power Brick.* The next most important thing is the thing we colloquially refer to as a "power brick."

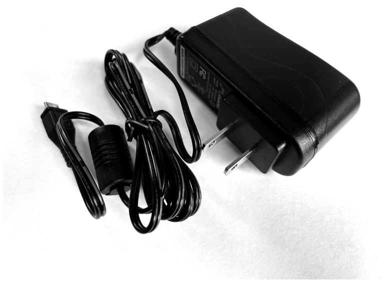


Figure 1.2: Power brick.

Digital logic circuits, like the ones found inside this computer, run on direct current. Power from a wall outlet is alternating current. This device converts power from a North American wall outlet (120 Volts AC) into the form used by this computer (5 Volts DC). Observe that the plug at one end of the power brick looks like a microUSB connector. That's because it *is* a microUSB connector. Like other inexpensive consumer electronics devices, the Raspberry Pi takes advantage of the ubiquity of USB chargers. If you lose your power brick, you can use whatever cellphone charger fits in the Raspberry Pi's power port.[1] Be careful not to plug your power brick into the wrong port—plug it into the port labeled on the Raspberry Pi's case as `Power`. Also observe that the power port has the text `PWR IN` printed next to it on the circuit board.

[1] If you travel outside North America with your Raspberry Pi to a location with a different power standard, just find a cellphone charger that works in your locality. If the charger has a microUSB connector, you can use it with your Raspberry Pi.

*Persistent storage.* The Raspberry Pi model we're using for this class lacks any persistent, built-in storage. Persistent storage saves the state of your machine when the power is turned off. You've probably heard people just say "disk" when they mean persistent storage. Typical computers use either an internal spinning magnetic disk (a "hard disk drive," aka "HDD") or an internal solid-state disk ("SSD"). We are using SD cards for this purpose, which are based on a similar technology as the SSDs found in most computers. SD cards have the advantage of being easily removable. The downside is that they are not very durable devices.

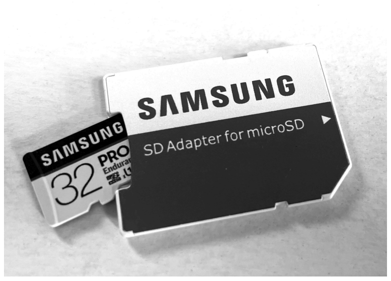


Figure 1.3: microSD card and SD adapter.

*USB SD card reader/writer.* We will eventually boot our Raspberry Pi computers using operating system software we have copied onto our SD card. This presents something of a chicken-and-egg problem. How do we put an operating system onto a disk when we need a running operating system to do so? The answer is to load the software from a different running machine.

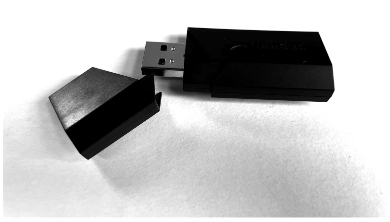


Figure 1.4: USB-to-SD card reader/writer.

This device plugs into any computer with a USB port. Note that you can't simply drag files from your computer's disk onto the icon that pops up when you plug this device into your computer. To boot successfully, operating system files need to be placed in specific locations, and all the files on the disk must be "formatted"[2] in a specific manner. Later in this lab, we will use a special program called ETCHER to write (sometimes called "flashing") files to our SD cards.

[2] We will be using the Linux operating system in this class. A typical Linux boot disk uses the EXT filesystem. If you are using a Mac, you're using either the HFS+ or APFS file formats. If you're using Windows, you're probably using NTFS. These formats are all very different and the differences matter.

*Serial console adapter.* Although you can connect a Raspberry Pi directly to a display with an HDMI port, a keyboard, and a mouse, most students nowadays use laptops and likely do not have spare displays, keyboard, and mice laying around. Instead, we will interface directly with the Raspberry Pi's serial port.

Watch out! Serial ports can be vulnerabilities or attack opportunities.

Serial ports are an ancient technology—they were originally invented to connect computers to teletype machines, which were large electromechanical machines that printed on paper. Printing on paper was important in early computing because *graphical* displays were both primitive and prohibitively expensive. Teletypes usually had a keyboard attached to them, which were familiar to many users who had worked with typewriters.

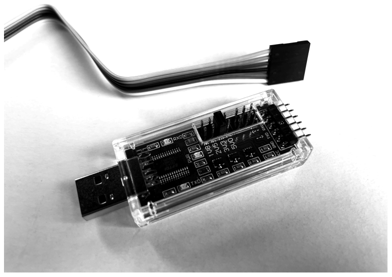


Figure 1.5: USB-to-serial console adapter.

Although we no longer use teletypes, the serial port stuck around because of its usefulness. Virtually every computer has a serial port, even if you can't see it.[3] When things go wrong in a computer system, the serial console almost always continues to operate without problems. It is not a coincidence that the text interface you learn as a computer science major is called "the terminal."[4]

[3] For example, even smartphones have something called a "JTAG port," which is essentially a serial port.

[4] The "Terminal" program in the macOS is actually a "virtual terminal," because it's only pretending to attach itself to the physical serial port on your computer.

A USB-to-serial console adapter lets you plug your Raspberry Pi's serial port into any computer with a USB port. As you will see in this lab, we are going to use the `screen` program to negotiate the connection between our two computers. This setup gives you the flexibility to physically use the computer of your choice (e.g., your laptop or a lab machine) while interacting with your Raspberry Pi. Note that the included ribbon cable makes the connection between the adapter and the Raspberry Pi's serial port pins.

*It is important that you connect the ribbon cable correctly. Failure to do so can damage the Raspberry Pi, the serial console adapter, or both.* We will discuss the procedure to follow in more detail later in this chapter.

*USB-C to USB-A adapter.* For those of you using late-model computers with USB-C ports, I've included a USB-C to USB-A adapter, since the two devices in our lab kit use USB-A.[5] If you don't have a USB-C port, you don't need to use this adapter.

Figure 1.6: USB-C to USB-A adapter.

[5] The astute reader will observe that this one lab kit includes *three* USB plugs. It sort of makes you wonder what the "U" in "USB" was supposed to stand for.

## 1.3 Step 1: Flash Your SD Card

*Flashing* a disk refers to process of cloning the contents of a *disk image* to a real disk. Real disks include SD cards, USB thumb drives, and hard disk drives. A *disk image* is a file representing a virtual disk. During flashing, the contents of a disk image are copied to a real disk. Commonly, disk image files have filenames ending in `.img`, `.iso`, or `.dmg`.

Before you can use your Raspberry Pi, you need to flash its SD card with an operating system. To make life easy, this course uses a disk image with a preconfigured copy of the Raspbian Linux operating system. [6] If you ever accidentally misconfigure or damage the software installed on your Raspberry Pi, you can follow this procedure to restore the disk back to a working state.

[6] Download from `https://csci331.s3.amazonaws.com/cs331_armhf_raspbian-2021-09-14.img.zip`.

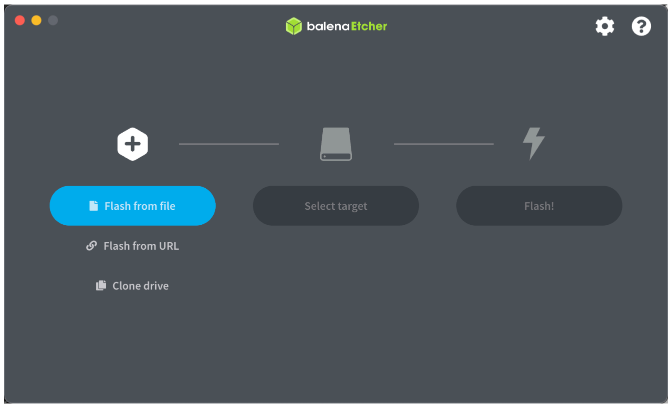
### 1.3.1 Flashing with Your Personal Computer

This section describes how to flash your SD card using a personal computer. Due to limitations in the Linux security model in a shared lab environment, this step cannot be carried out on a CS lab machine. If you do not have a personal computer you can use, let me know, and I will supply you with a pre-imaged SD card.

1. Insert your microSD card into your USB SD card reader/writer. You do not have to use the microSD adapter, but you can if you want to.

2. Plug the adapter into your computer. Use the USB-C to USB-A adapter if your computer lacks a USB-A socket.

3. Next, we use a program called Etcher[7]. After installing it, look for a graphical program called `balenaEtcher`. Starting it should produce a screen that looks something like the following, depending on your operating system.



4. Click on the button labeled `Flash from file`, and when prompted, select the image you downloaded earlier (`cs331_armhf_raspbian-2021-09-14.img.zip`).

[7] Download Etcher from `https://www.balena.io/etcher/`. Etcher is available for the macOS, Windows, and Linux. If you're brazenly running a weirdo OS, like Solaris, OpenBSD, or Haiku, you can use `dd`, an exercise left to the reader. Adventurous readers can use `dd` on non-weirdo OSes too.

5. Click on the button labeled `Select target`, and when prompted, check the box belonging to your SD card reader/writer. The adapter is called "SABRENT SD Media" on my machine. Importantly, the capacity of the disk should be roughly 32GB.

   Be very careful not to select a different disk. Flashing overwrites the contents of whatever disk you choose. If you're not careful, you can accidentally destroy personal data!

   Click the `Select(1)` button.

6. Now click the `Flash!` button. Flashing will take several minutes.

7. When Etcher finishes, quit the program and remove the microSD card from the adapter.

## 1.4 *Step 2: Connect a Serial Console Adapter to Your Computer*

In this step, you are going to connect the serial console adapter to your computer (e.g., your laptop or lab machine). We will not connect the adapter to the Raspberry Pi just yet. The purpose of this step is to ensure that your computer can communicate with the adapter. To avoid confusion, let's call the computer we're connecting to adapter to the *host computer*. Remember, the host computer is *your* machine (e.g,. your laptop), not the Raspberry Pi.

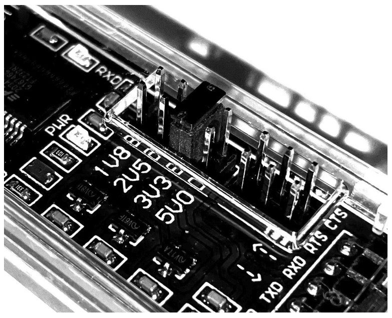


Figure 1.7: Top jumper set to 3V3.

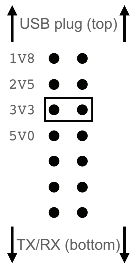


Figure 1.8: Top jumper set to 3V3 (schematic).

Fresh out of the box, your serial console adapter will come with two *jumpers* installed. A jumper is a small conductor that connects one electrical contact to another. The first jumper, on the top of the serial console adapter, sets the voltage of the adapter. Your Raspberry Pi uses 3V signals, so your jumper should be set to the `3V3` setting. Figure 1.12 shows a correctly set jumper, and Figure 1.8 shows the same setting in schematic form.

Setting your serial console adapter to the wrong voltage can damage your Raspberry Pi. Do this step carefully!

Go ahead and set the top jumper to `3V3`.

The second jumper is at the bottom of your serial console adapter. Note that the pins at the bottom of the adapter are labeled `VCC`, `GND`, `TXD`, `RXD`, `RTS`, and `CTS` from left to right. Be sure that the bottom jumper *jumps* the TXD and RXD pins on your own adapter. Figure 1.9 shows the correct placement of the bottom jumper. Set that jumper now.

In normal operation, we will remove the bottom jumper. However, since we want to make sure that our adapter works, the bottom jumper makes something special happen: data transmitted over the adapter will be *echoed* back to us. This works because data sent from the adapter is sent as a signal on the TXD pin. TXD stands for "transmit data." Data is received by the adapter on the RXD pin, which stands for "receive data". Since we bridged the two pins with a jumper, which electrically connects the two, data sent on TXD is immediately sent back on RXD.

Now that we have configured our serial console adapter correctly, plug it into a USB port on the host computer. If your computer has a USB-C port, use the supplied USB-C to USB-A adapter. Once the adapter is connected, a red light labeled PWR will become illuminated.

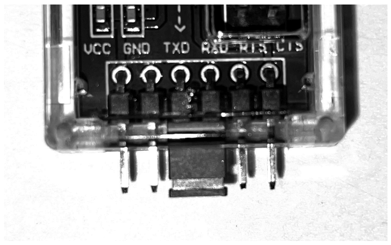


Figure 1.9: Bottom jumper set to TXD/RXD.

## 1.5 *Step 3: Start a Console Emulator on the Host Computer*

If you are a Windows user, skip ahead to the section titled *On Windows*.

For a variety of reasons, your device name may be different. If this happens, you will need to find the name on your own computer. One way to do it is to type `$ ls /dev/tty*` before plugging in the adapter, plug it in, run `$ ls /dev/tty*` again, and look for a new name in the output. If you have a an account with superuser privileges (e.g., you are the owner of the machine), you can also inspect the output of running `$ sudo dmesg`, which sometimes prints the name of the device when it is plugged into the computer.

To make the host computer communicate over the serial console adapter, we will run a program called `screen`. GNU `screen` is a *virtual console multiplexor*, meaning that

- it pretends to be a serial console, *and*
- it allows you to display multiple virtual consoles in a single window.

We're going to focus on the first item, but `screen` is a useful tool for juggling multiple consoles (whether or not you use serial adapters), and so if you're curious, it's worth looking at the documentation that comes up when you type `man screen` at the prompt. For now, let's use `screen` to connect to your serial console adapter.

*On a Mac.* Open a Terminal window and type the following.

```
$ screen /dev/tty.usbserial-[something] 115200
```

where `[something]` is whatever appears when you type in

```
$ ls /dev/tty.usbserial-*
```

after plugging in the adapter.

Baud, which measures the number of symbols per second, was named after Émile Baudot, inventor of a widely used data encoding for telegraph systems in the 19th century. Since, like telegraphs, serial consoles send text, the unit stuck.

*On Linux.* Open a Terminal window and type the following.

```
$ screen /dev/ttyUSB0 115200
```

The first part of our `screen` command says which *device name* to connect `screen` to.

Device names are different on the Mac versus Linux, which is why these commands are slightly different. The second part of our `screen` command says *what speed* we should run our adapter. Speed uses a unit called *baud*. We're telling `screen` to send data at 115,200 baud.

You can quit screen at any time by typing ctrl + a, ctrl + \\.

*On Windows.* Windows users need both a driver for the USB-TTY device we are using in class, and a third-party terminal emulator, as `screen` is not available. Follow these steps to install both.

1. Download the USB-TTY driver. [8]

[8] Download from `https://csci331.s3.amazonaws.com/CDM212364_Setup.zip`.

2. Right-click on the `CDM212364_Setup.zip` file and select `Extract All...`

3. Click the `Extract` button on the window that comes up.

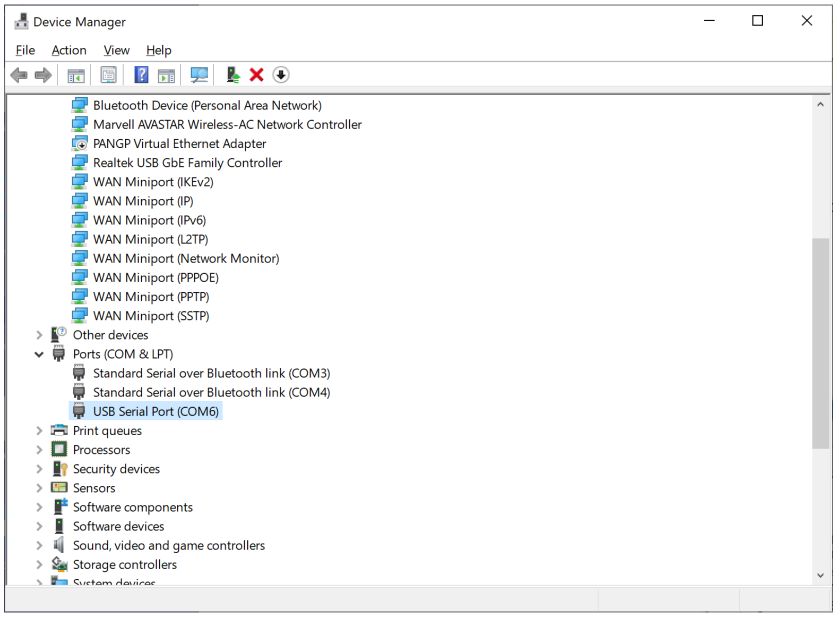
4. In your file explorer, find the file that you just extracted, `CDM212364_Setup.exe`. Double-click on the executable and follow the instructions to install it. You will need to reboot your computer.

5. Download the PuTTY installer. [9]

[9] Download from `https://csci331.s3.amazonaws.com/putty-64bit-0.76-installer.msi`.

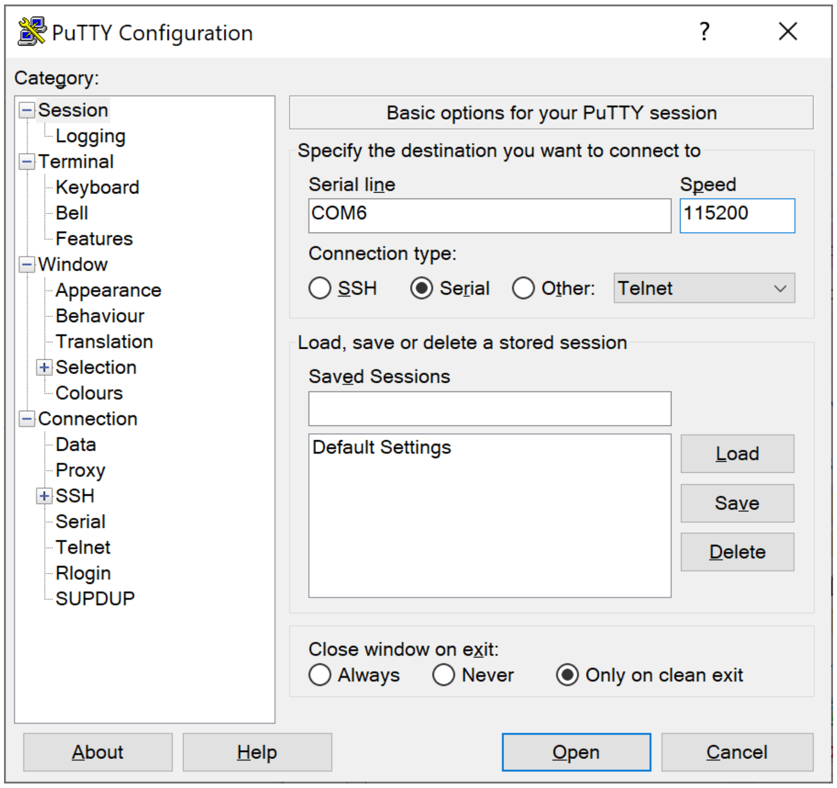
6. Double-click on the installer and follow the directions.

If your serial console adapter is plugged in and you installed the driver correctly as above, Windows will assign a "`COM` port" to the device. To discover the `COM` port mapping, go to your Start menu and type "`device manager`" and when the `Device Manager` program appears, start it. Look for the `Ports (COM & LPT)` item and expand it. There should be an entry called `USB Serial Port`, and in parentheses, it will say what the `COM` port mapping is. On my machine, the serial console adapter maps to `COM6`.



```
        Bluetooth Device (Personal Area Network)
        Marvell AVASTAR Wireless-AC Network Controller
        PANGP Virtual Ethernet Adapter
        Realtek USB GbE Family Controller
        WAN Miniport (IKEv2)
        WAN Miniport (IP)
        WAN Miniport (IPv6)
        WAN Miniport (L2TP)
        WAN Miniport (Network Monitor)
        WAN Miniport (PPPOE)
        WAN Miniport (PPTP)
        WAN Miniport (SSTP)
  >   Other devices
  v   Ports (COM & LPT)
        Standard Serial over Bluetooth link (COM3)
        Standard Serial over Bluetooth link (COM4)
        USB Serial Port (COM6)
  >   Print queues
  >   Processors
  >   Security devices
  >   Sensors
  >   Software components
  >   Software devices
  >   Sound, video and game controllers
  >   Storage controllers
  >   System devices
```

Find the PuTTY program in your Windows Start menu and start it. You will be presented with the following dialog box.



Specify the destination you want to connect to

Serial line: COM6

Speed: 115200

Connection type: SSH [✓ Serial] Other: Telnet

Load, save or delete a stored session

Saved Sessions

Default Settings

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

Under `Connection type`, choose `Serial`, in the `Serial line` field that appears, enter `COM`*n* (where *n* is the mapping you discovered earlier), and in the `Speed` field, enter `115200`.

Finally, click the `Open` button, and a blank console will appear.

## 1.6 *Step 4: Observe the Blinkenlights*

Once your terminal emulator is running, try typing. You should notice two things. First, when you type, two lights, labeled `TXD` and `RXD` on your serial console adapter, should flash. Second, you should see text appear on your screen.

To demonstrate that the "text echo" you see in `screen` really is because you are reflecting the `TXD` signal into `RXD`, remove the jumper you put on the `TXD`/`RXD` pins in the previous step. Now when you type, you should see the `TXD` light flash, but not the `RXD` light, and no text will appear in your `screen` window. Once you're satisfied that you understand what is happening, quit your terminal emulator. You can quit `screen` by typing ctrl + a, ctrl + \. You can quit `PuTTY` by closing the program. Then remove the serial console adapter from the host computer's serial port, and leave the `TXD`/`RXD` jumper off.

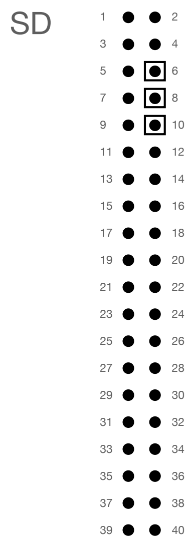


Figure 1.10: GPIO pin numbers. When oriented so that the Raspberry Pi's SD card slot is up, pin **1** is at the top left. `GND` = pin 6, `TXD` = pin 8, `RXD` = pin 10.

## 1.7 *Step 5: Connect a Serial Console Adapter to the Raspberry Pi*

With the serial console adapter *disconnected* from your computer, attach the flat, wide part of the ribbon cable to your adapter. Note that the ribbon cable can be attached in one of two orientations, with either the white wire attached to the pin marked `VCC` or with the white wire attached to the pin marked `CTS`. Although the color of an electrical wire typically means something, there is no way to insert this cable so that the colors align with their traditional meanings[10]. Therefore, we're going to choose an arbitrary orientation.

[10] Traditionally, the "hot wire" is black, the "neutral wire" is white, and the "ground" wire is green

On the other end of the ribbon cable, you should see individual plugs. Connect each plug wire to its corresponding pin on the Raspberry Pi. The table below shows how pins should be connected.

| Adapter Pin | Color | RPi Pin | Purpose |
|---|---|---|---|
| `CTS` | orange | *n*/*a* | Flow control (not used). |
| `RTS` | green | *n*/*a* | Flow control (not used). |
| `RXD` | blue | 8 `TXD` | Receive data from RPi. |
| `TXD` | black | 10 `RXD` | Send data to RPi. |
| `GND` | red | 6 `GND` | Ground. |
| `VCC` | white | *n*/*a* | Power supply (not used). |

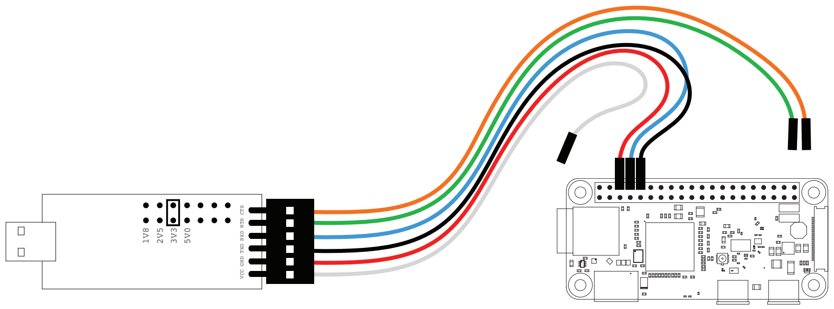


Figure 1.11: Connect the flat end of the ribbon cable to the serial console adapter. Connect wires on the plug end to the `GND`, `RXD`, and `TXD` pins on the Raspberry Pi.

## 1.8 *Step 6: Insert microSD Card and Power Up*

With your Raspberry Pi powered off, insert the microSD card you flashed earlier into the microSD card slot. Be sure that your serial console adapter is plugged into your host computer's USB port and that it is receiving power. Now start up the `screen` program again as we did before. This time, connect the USB 5V power supply to the port on the Raspberry Pi labeled `PWR` (or on the board, labeled `PWR IN`). If you did all the steps correctly, you should see text like the following appear on your console screen as your Raspberry Pi boots up.

```
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 5.10.17+ (dom@buildbot) (arm-linux-
    gnueabihf-gcc-8 (Ubuntu/Linaro 8.4.0-3ubuntu1) 8.4.0, GNU ld
     (GNU Binutils for Ubuntu) 2.34) #1414 Fri Apr 30 13:16:27
    BST 2021
[    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] OF: fdt: Machine model: Raspberry Pi Zero W Rev
    1.1
[    0.000000] Memory policy: Data cache writeback
[    0.000000] Reserved memory: created CMA memory pool at 0
    x17c00000, size 64 MiB
[    0.000000] OF: reserved mem: initialized node linux,cma,
    compatible id shared-dma-pool
[    0.000000] Zone ranges:
[    0.000000]   Normal   [mem 0x0000000000000000-0
    x000000001bffffff]
(and so on...)
```

Once the Raspberry Pi is done booting, it will print a login prompt.

```
Raspbian GNU/Linux 10 raspberrypi ttyS0

raspberrypi login:
```

To login, type the username **`pi`** and password **`raspberry`**.

If you wish, change your password using the `passwd` utility:

```
  $ passwd
```

This class uses `Raspbian`, version `10 (``buster'')`. You can learn more about `Raspbian` at `https://www.raspberrypi.org/software/operating-systems/`.

When you are done using your Raspberry Pi, it is very important that you perform a *clean shutdown*. A clean shutdown is when you run the system `shutdown` script to halt the computer. This script does a number of important bookkeeping tasks, including ensuring that all pending writes to disk are written out, and that the disk's metadata journal is consistent with these writes. Failure to shutdown cleanly risks data loss or system corruption.

## 1.9 *Step 7: Do a clean shutdown*

Run the following command, which will turn your Raspberry Pi off.

```
$ sudo shutdown -h now
```

Note that you can also reboot your computer using the following.

```
$ sudo shutdown -r now
```

After issuing the shutdown command, be sure to wait until the green `PWR` LED on the Raspberry Pi has gone out. The system has not completely powered down until the light is off. Then, pull the power plug and re-insert it, which will boot the computer again.

The `sudo` command temporarily gives your user account superuser privileges. The name is derived from the phrase "superuser do," and so it should be correctly pronounced like "sue doo." Nevertheless, many hackers are autodidacts and therefore most of them pronounces it "sue dough." Choose the pronunciation that spares you the most embarrassment in social situations. Either way, `sudo` is an extremely powerful tool that gives you the power to do *anything* on your computer. Use it carefully!

## 1.10 *Step 8: Configure Console Dimensions*

The serial console standard, called RS-232, is old compared to most computer technologies, originally introduced in 1960. RS-232 predates modern graphical window environments by many years.

When you run the `Terminal` program on your Mac, the `PuTTY` program on your Windows machine, or the `Konsole` on your Linux machine, what you are actually running is a kind of program called a *terminal emulator*, often called a *term* for short. In other words, it's a program that emulates a serial console. Since RS-232 knows nothing about this, both ends of the console session make some assumptions about the *dimensions* of your window. As the earliest serial consoles were always textual, dimensions are in terms of *rows* and *columns* of text. The default for many serial consoles, including ours, is 24 rows by 40 columns, and by default, your terminal window is likely to be small.

Unfortunately, when you resize your terminal window, you'll probably discover that the text inside it does not change dimensions. RS-232 is not capable of communicating window dimensions. Instead, both endpoints need to agree ahead of time. Fortunately, telling your Raspberry Pi how many rows and columns are being used is easy.

Figure 1.12: Your computer's terminal emulator still thinks it is one of these. Photo © 2013 Jason Scott.

1. Resize your graphical window by clicking on one of the corners and dragging.

2. Find your window size measurement.

   (a) *macOS*: Look for a pair of numbers, like `154x90` printed in the `Terminal`'s titlebar text. The first number is the number of *columns* and the second number is the number of *rows*.

   (b) *Linux*: To obtain the number of rows,

   ```
   $ tput lines
   ```

   To obtain the number of columns,

   ```
   $ tput cols
   ```

   (c) *Windows*: In `PuTTY`, right-click on the titlebar and select "Change Settings...", then find "Window" in the menu that appears. The dimensions will be listed on this configuration page.

3. In your `screen` session on your Raspberry Pi, use the following template:

   ```
   $ stty rows <rows> columns <columns>
   ```

   For example, since my `Terminal` window says `154x90`, I would type:

   ```
   $ stty rows 90 columns 154
   ```

Now, your serial console will span the entire width and height of your terminal emulator window. Remember that whenever you resize the window, you will need to repeat this process again.

## 1.11 *Step 9: Configure Wifi*

By default, your Raspberry Pi is not connected to any computer networks. Since a computer without a network connection is limited in usefulness, you are going to connect your Raspberry Pi to the campus `eduroam` network.

1. Generate a *hashed* version of your `eduroam` password. Type:

   ```
   $ echo -n 'password_in_plaintext' | iconv -t utf16le | openssl md4
   ```

   where `password_in_plaintext` is substituted with your campus login's password. Note that you *must* enclose your password in single quote characters (`'`). Also note that the last two characters of `utf16le` are the lowercase letters `L` and `E`. You should see a long, hexadecimal number printed on your console, something like.

   ```
   (stdin)= 8265ff84873220f65fb54e6beae931b7
   ```

Copy this number.

2. Use the nano editor to edit the file `/etc/wpa_supplicant/wpa_supplicant.conf`.

```
$ sudo nano -w /etc/wpa_supplicant/wpa_supplicant.conf
```

3. When nano starts up, you should see the following:

```
network={
      ssid="eduroam"
      priority=1
      proto=RSN
      key_mgmt=WPA-EAP
      pairwise=CCMP
      auth_alg=OPEN
      eap=PEAP
      identity="username"
      password=hash:password
      phase1="peaplabel=0"
      phase2="auth=MSCHAPV2"
}
```

Substitute `username` and `password` with their appropriate values. For example, I would put `dwb1` in the `identity` field and `hash:8268ae84873222f65fbb8e6be11931b1` in the `password` field.

4. Press ctrl + o to save the document.
5. Press ctrl + x to quit nano.
6. Use the nano editor to edit the file called `/etc/network/interfaces`

```
$ sudo nano -w /etc/network/interfaces
```

7. When nano starts up, ensure that the document contains the following:

```
auto lo

iface lo inet loopback

allow-hotplug wlan0

iface wlan0 inet dhcp
        pre-up wpa_supplicant -B -Dwext -i wlan0 -c/etc/wpa_supplicant/wpa_supplicant.conf
        post-down killall -q wpa_supplicant
```

8. Save and quit nano as you did before.
9. Reboot your Raspberry Pi.

```
$ sudo shutdown -r now
```

10. After your computer has rebooted, verify that it has connected to `eduroam` using the `ifconfig` tool. Type

```
$ ifconfig
```

In `ifconfig`'s output, you should see something like the following.

```
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 137.165.126.163  netmask 255.255.248.0  broadcast 137.165.127.255
        inet6 fe80::ba27:ebff:febc:d6c9  prefixlen 64  scopeid 0x20<link>
        ether b8:27:eb:bc:d6:c9  txqueuelen 1000  (Ethernet)
        RX packets 30  bytes 7550 (7.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 47  bytes 6649 (6.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Be sure that you are looking at the block for `wlan0` and not `lo`. `wlan0` is your wireless network interface card (NIC), while `lo` is a "mock" network device referred to as "loopback" that your computer uses to communicate with itself. In the `wlan0` section, look for an IP address, right after the label `inet`. Mine here is `137.165.126.163`. If you see a number of the form `69.254.xxx.xxx`, then you have not configured your wireless correctly. On the Williams campus, your address will almost always start with `137`. Repeat the steps above and look for your mistake.

## 1.12 *Step 10: Install Some Software*

The `Raspbian` operating system is a variant of the Debian Linux operating system. Debian uses a tool called `apt` to manage software installation. Let's use `apt` to install some software.
First we need to update `apt`'s catalog of packages. Run

```
  $ sudo apt update
```

This process will take a few minutes, and at the end you should see output like:

```
Get:1 http://archive.raspberrypi.org/debian buster InRelease [32.6 kB]
Get:2 http://raspbian.raspberrypi.org/raspbian buster InRelease [15.0 kB]
Get:3 http://archive.raspberrypi.org/debian buster/main armhf Packages [380 kB]
Get:4 http://raspbian.raspberrypi.org/raspbian buster/main armhf Packages [13.0 MB]
Fetched 13.4 MB in 28s (478 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Let's install a few tools you're going to need for basic programming, the `gcc` compiler and the `git` version control system.

```
  $ sudo apt install gcc git
```

If you have favorite (non-graphical) applications, go ahead and try installing them with `apt`.

As of the writing of this document, the `emacs` editor does not correctly recognize certain control characters when used with the serial console. I suggest using the `nano` editor instead, which is already installed.

## 1.13 *Step 11: Have a Little Fun: Network Scanning*

We'll conclude this lab by doing some network detective work. The tools we install below are a major component of network vulnerability scanning and defense planning. Because network security is a deep topic, we can't really do it justice in this class. However, the activity below should perhaps whet your whistle.

Let's start by installing `whois`.

```
$ sudo apt install whois
```

After it's installed, try it out. Who owns the address `26.0.0.113`? Recall that this was an address Clifford Stoll mentioned in *The Cuckoo's Egg*.

```
$ whois 26.0.0.113
```

You should see the `WHOIS` entry for the `DISANET26` network.

Let's install `traceroute`. The `traceroute` tool tells us the network path data takes from our machine to a given host.

```
$ sudo apt install traceroute
```

After it's installed, try it out.

```
$ traceroute 26.0.0.113
```

Finally, let's install the network exploration tool, `nmap`.

```
$ sudo apt install nmap
```

Let's scan our local network. Find your IP address using the `ifconfig` tool. For example, mine is `137.165.126.163`. My computer is but one connected machine from among many connected machines on the network. Networks are organized into *netblocks*. A netblock is what it sounds like: a collection of network addresses.

Netblocks are typically given to an organization in chunks that share a prefix. For example, since my address is `137.165.126.163`, it's likely that there is another host on the network that shares the `137.165.126` prefix. The most common prefix for small networks is what is called a "class C network." In other words, the first three numbers match. We use a notation called *classless interdomain routing*, or CIDR, to describe netblock prefixes. For example, if our host is on a network that shares the first three numbers, the CIDR notation for that network will be `137.165.126.163/24`.

CIDR is usually pronounced "cider."

CIDR notation has the following form: `<prefix>/<netmask>`. The `<prefix>` is just a network address. The `<netmask>` is a number between 1 and 31 that describes how many *bits* of the address are *masked*, from right to left. A masked number is "fixed," meaning that all hosts in the given CIDR network share that part. The unmasked component is "variable," and describes all of the possible addresses that a host on that network might have.

An IPv4 network address always has four numbers. Each number is represented using 8 bits. Therefore, an IPv4 address is represented using 32 bits. Returning to our example, `137.165.126.163/24` means

Network operators sometimes call IPv4 addresses "dotted quads." There are other addressing schemes for IP networks, but for this example, we'll stick with IPv4, since it is the most common.

that the first 24 bits are fixed. That means that all hosts on that network must share the first three numbers $(3 \times 8 = 24)$, `137.165.126`, and the last number can be anything between 0 and 255. In practice, 0 and 255 have special meanings, so `137.165.126.163/24` spans hosts `137.165.126.1`—`137.165.126.254`. In recognition that the last IPv4 number in this CIDR network does not matter, networks like this will sometimes be written `137.165.126.0/24` or `137.165.126/24`.

Let's scan `137.165.126.0/24`. Note that `nmap` requires you to write out four IP digits in your CIDR network or it does not understand what you mean.

Security-conscious organizations are generally wary of network scans and frown on them, because the number of users who legitimately need to perform them is small, usually just technical staff. Unauthorized scans often mean trouble. For this class, we have been given permission by Williams' head of IT to do these scans. Be aware that scanning the network contacts hosts on that network, meaning that your computer will be quite visibly scanning the network. If you run `nmap` outside of Williams, you may very well be contacted by an irate network administrator or simply have your network access summarily terminated. You *definitely* do not want to scan the military address listed above!

```
  $ nmap -sn 137.165.126.0/24
```

You should see some output like the following:

```
$ nmap -sn 137.165.126.0/24
Starting Nmap 7.70 ( https://nmap.org ) at 2021-09-06 20:32 BST
Nmap scan report for 137.165.126.44
Host is up (0.013s latency).
Nmap scan report for 137.165.126.66
Host is up (0.10s latency).
Nmap scan report for 137.165.126.68
Host is up (0.021s latency).
Nmap scan report for 137.165.126.69
Host is up (0.017s latency).
...
```

Each host listed at the given address is connected to the network. Now let's see what network applications (i.e., "services") those hosts expose to the network. This will take a few minutes.

```
  $ nmap 137.165.126.0/24
```

```
Starting Nmap 7.70 ( https://nmap.org ) at 2021-09-06 20:34 BST
Nmap scan report for 137.165.126.44
Host is up (0.0076s latency).
All 1000 scanned ports on 137.165.126.44 are closed

Nmap scan report for 137.165.126.66
Host is up (0.0070s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh

Nmap scan report for 137.165.126.68
Host is up (0.0070s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh

Nmap scan report for 137.165.126.69
Host is up (0.0085s latency).
Not shown: 999 closed ports
PORT      STATE SERVICE
62078/tcp open  iphone-sync
...
```

When you are done, you should see a list of hosts and their services. For example, in the output above, we can see that `137.165.126.66` is running the secure shell program, `ssh` on port 22.

What netblock does Williams *really* use? I told you to scan `137.165.126.0/24`, but that's not actually the entire Williams network. Try using the `whois` tool to find their allocated netblock in CIDR notation.

To see what other things `nmap` can do, like *fingerprinting* hosts, see

```
$ man nmap
```

When you are done, don't forget to *cleanly shutdown* your computer.