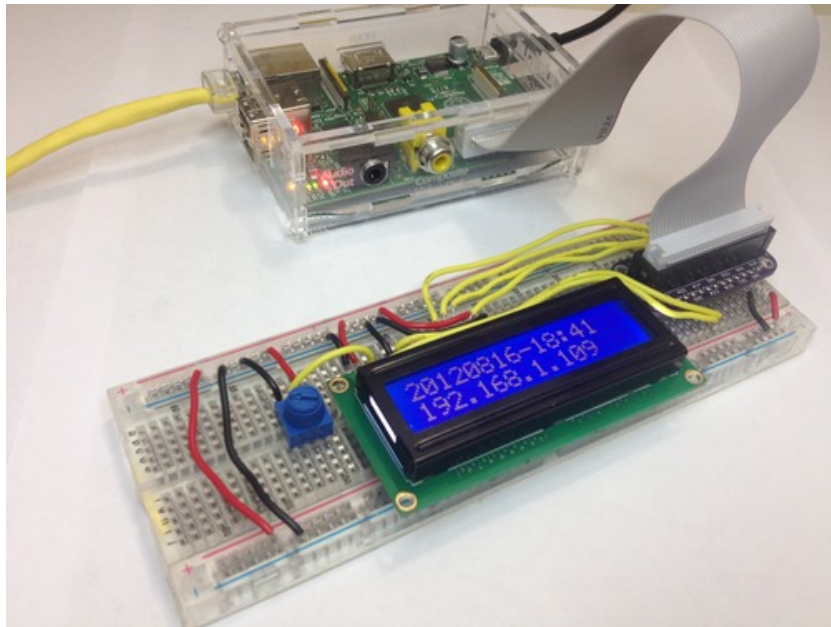




Drive a 16x2 LCD with the Raspberry Pi

Created by Mikey Sklar

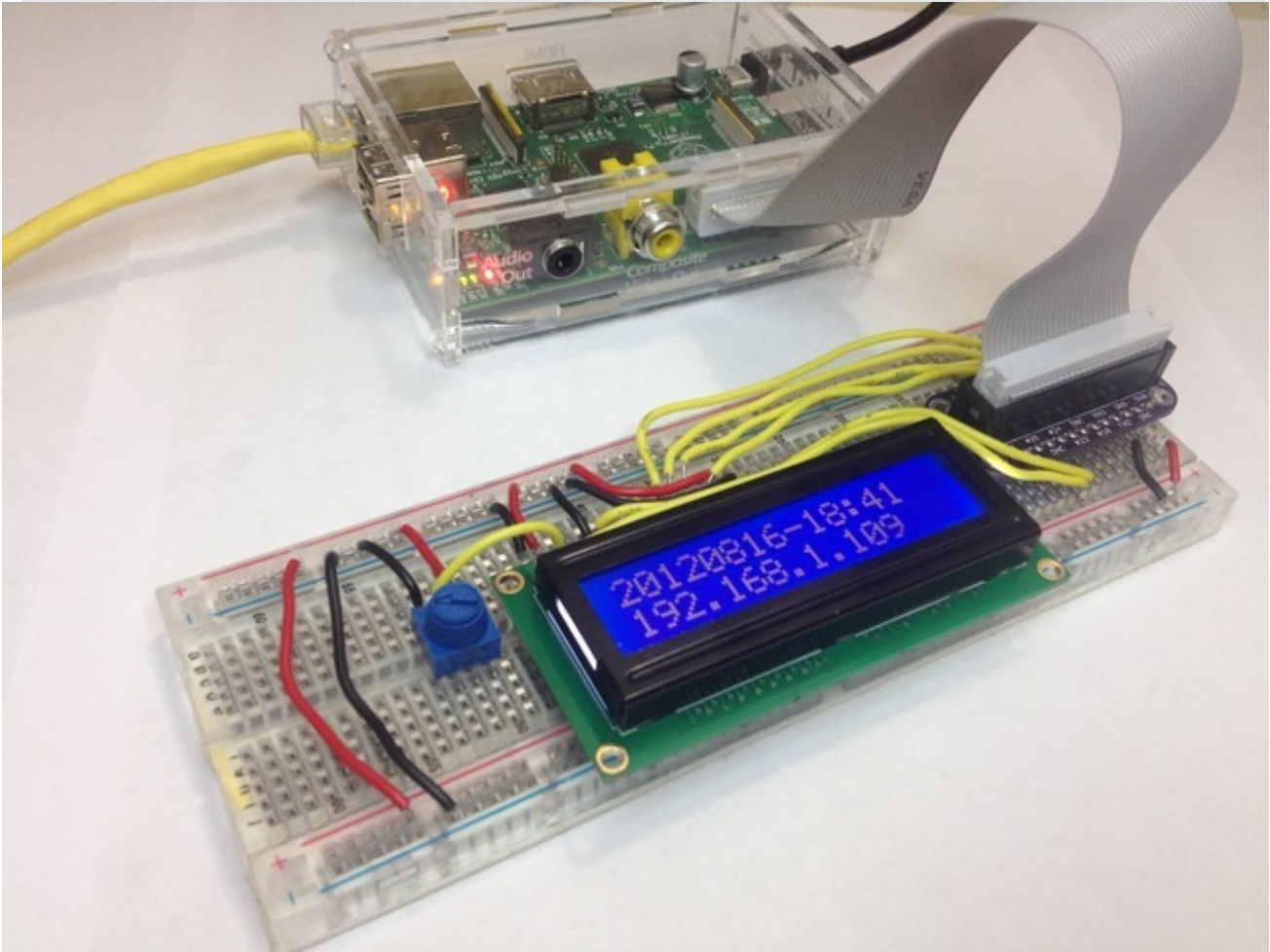


Last updated on 2014-07-24 04:15:37 AM EDT

Guide Contents

Guide Contents	2
Overview	3
To Follow This Tutorial You Will Need	4
Wiring the Cobbler to the LCD	5
The LCD	5
LCD Pinout	6
Wiring Diagram	6
Schematic	7
5v LCD vs 3.3v Pi	8
Preparing the LCD	8
Necessary Packages	15
Python Script	17
The Code	17
Testing	17
IP Clock Example	18
Running the Code	18
Start the IP + Clock example	19
What You Should See	19
Init Script	20
Time Zone	22

Overview

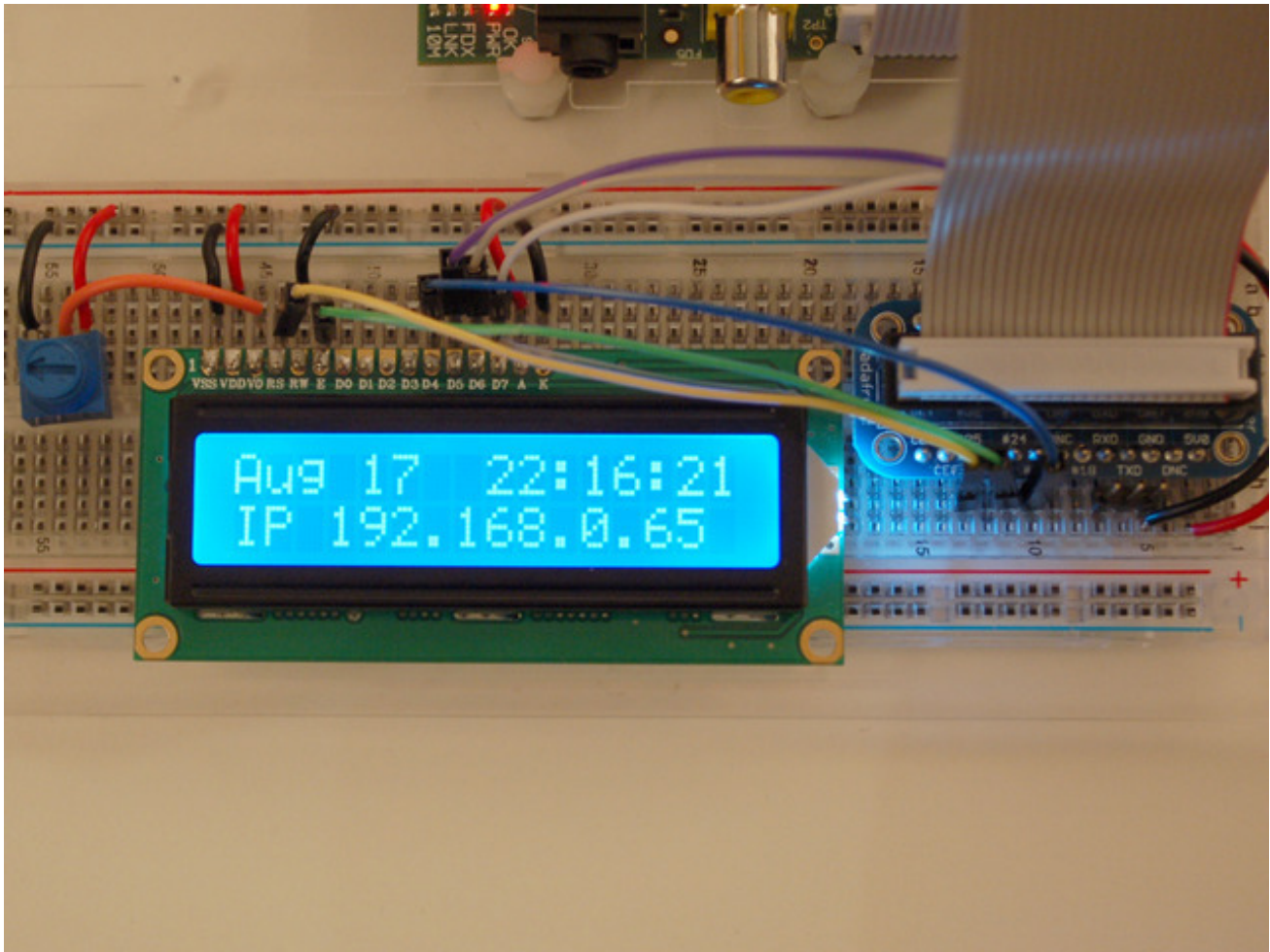


Adding a LCD to any project immediately kicks it up a notch. This tutorial explains how to connect a inexpensive HDD44780 compatible LCD to the raspberry pi using 6 GPIOs. While there are other ways to connect using I2C or the UART this is the most direct method that get right down to the bare metal.

This technique:

- allows for **inexpensive LCDs** to be used
- does not require any **i2c drivers**
- won't steal the only **serial port** on the Pi.

The example python code sends date, time and the ip address to the display. If you are running a Pi in headless mode being able to determine the IP address at a glance is really handy.



To Follow This Tutorial You Will Need

- Standard LCD 16x2 + extras (<http://adafruit.it/181>)
- Adafruit Pi Cobbler (<http://adafruit.it/914>) - follow the tutorial to assemble it
- Half (<http://adafruit.it/64>) or Full-size breadboard (<http://adafruit.it/239>)
- Hook-up Wire (<http://adafruit.it/aM5>)
- A Raspberry Pi

You can use nearly any character LCD with this tutorial - it will work with 16x1, 16x2, 20x2, 20x4 LCDs. It will not work with 40x4 LCDs

Wiring the Cobbler to the LCD

The LCD



Whenever you come across a LCD that looks like it has 16 connectors it is most likely using a HD44780 controller. These devices provide the same pinouts making them relatively easy to work with. The LCD uses a parallel interface meaning that we will need many pins from our raspberry pi to control it. In this tutorial we will use 4 data pins (4-bit mode) and two control pins.

The **data pins** are straight forward. They are sending data to the display (toggled high/low). We will only be using write mode and not reading any data.

The **register select** pin has two uses. When pulled low it can send commands to the LCD (like position to move to, or clear the screen). This is referred to as writing to the instruction or command register. When toggled the other way (1) the register select pin goes into a data mode and will be used to send data to the screen.

The **read/write** pin will be pulled low (write only) as we only want to write to the LCD based

on this setup.

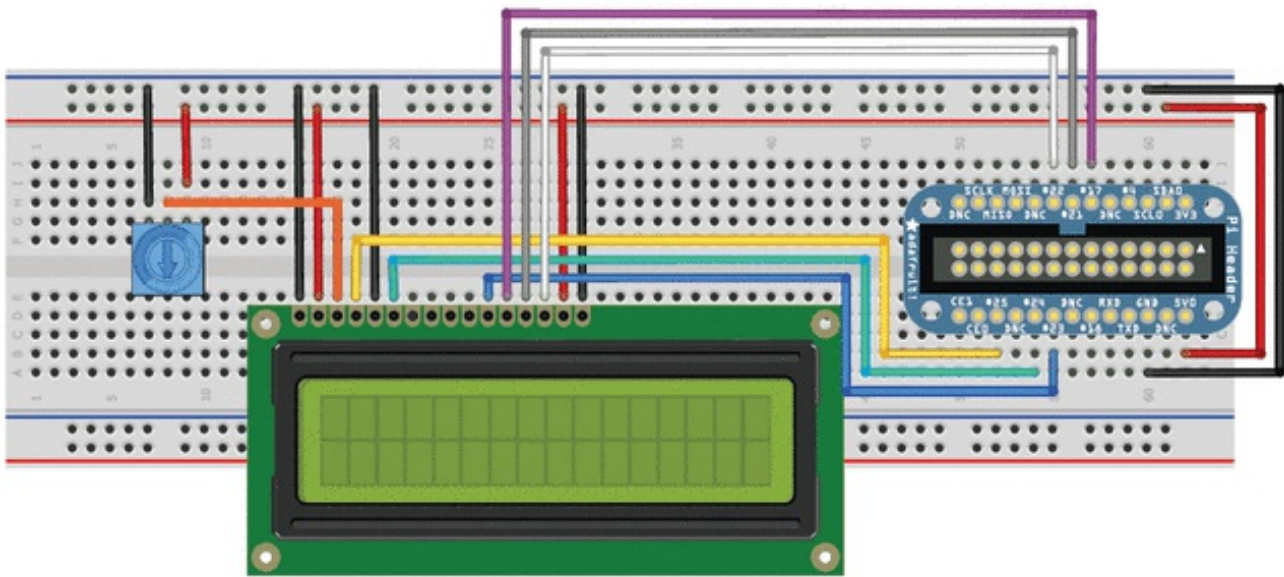
The **enable** pin will be toggled to write data to the registers.

LCD Pinout

1. Ground
2. VCC - **5v not 3.3v**
3. Contrast adjustment (VO) from potentiometer
4. Register Select (RS). RS=0: Command, RS=1: Data
5. Read/Write (R/W). R/W=0: Write, R/W=1: Read (**we won't use this pin**)
6. Clock (Enable). Falling edge triggered
7. Bit 0 (**Not used in 4-bit operation**)
8. Bit 1 (**Not used in 4-bit operation**)
9. Bit 2 (**Not used in 4-bit operation**)
10. Bit 3 (**Not used in 4-bit operation**)
11. Bit 4
12. Bit 5
13. Bit 6
14. Bit 7
15. Backlight LED Anode (+)
16. Backlight LED Cathode (-)

Before wiring, check that your LCD has an LED backlight, not an EL backlight. LED backlights use 10-40mA of power, EL backlights use 200+ma! EL backlights are often cheap to get but are not usable, make sure you don't use one or you will overload the Pi. Some cheap LCDs that have LED backlights do not include a resistor on the LCD module for the backlight, if you're not sure, connect a 1Kohm resistor between pin 15 and 5V instead of connecting directly. **All Adafruit LCDs have LED backlights with built in resistors so you do not need an extra resistor!**

Wiring Diagram



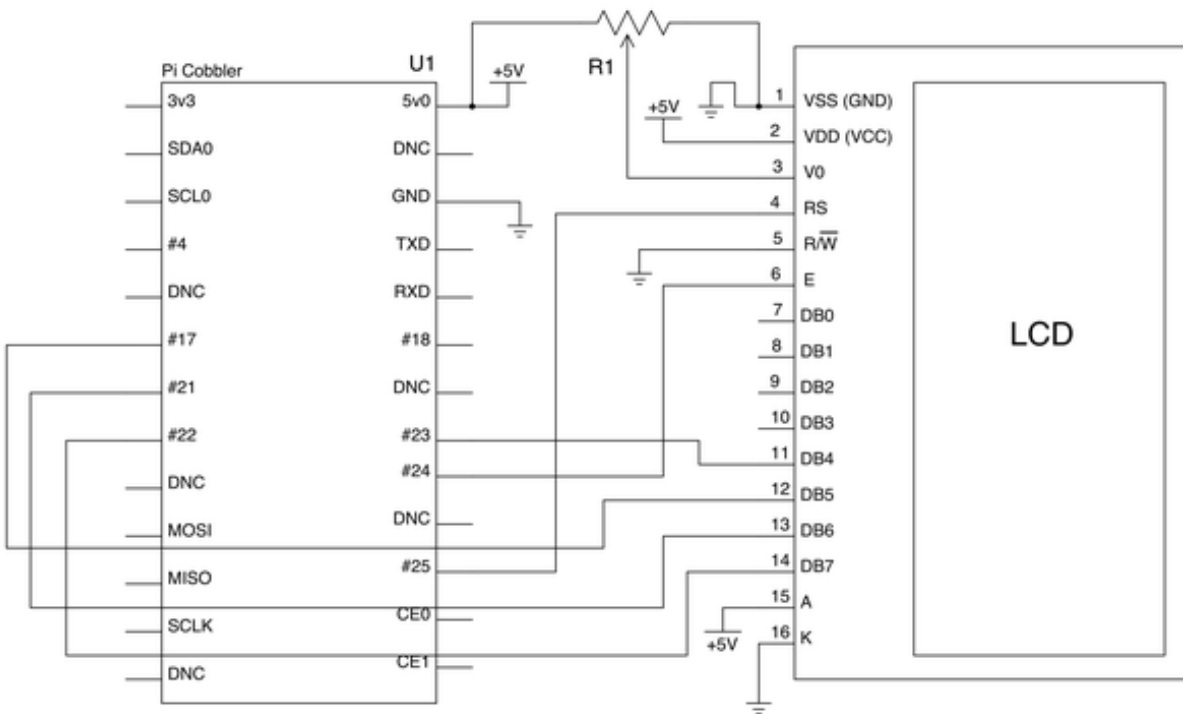
First, connect the cobber power pins to the breadboard power rail. +5.0V from the cobber goes to the red striped rail (red wire) and GND from the cobber goes to the blue striped rail (black wire)

In order to send data to the LCD we are going to wire it up as follows

- Pin #1 of the LCD goes to ground (black wire)
- Pin #2 of the LCD goes to +5V (red wire)
- Pin #3 (Vo) connects to the middle of the potentiometer (orange wire)
- Pin #4 (RS) connects to the Cobber #25 (yellow wire)
- Pin #5 (RW) goes to ground (black wire)
- Pin #6 (EN) connects to cobber #24 (green wire)
- Skip LCD Pins #7, #8, #9 and #10
- Pin #11 (D4) connects to cobber #23 (blue wire)
- Pin #12 (D5) connects to cobber #17 (violet wire)
- Pin #13 (D6) connects to cobber #21 (gray wire)
- Pin #14 (D7) connects to cobber #22 (white wire)
- Pin #15 (LED +) goes to +5V (red wire)
- Pin #16 (LED -) goes to ground (black wire)

Then connect up the potentiometer, the left pin connects to ground (black wire) and the right pin connects to +5V (red wire)

Schematic



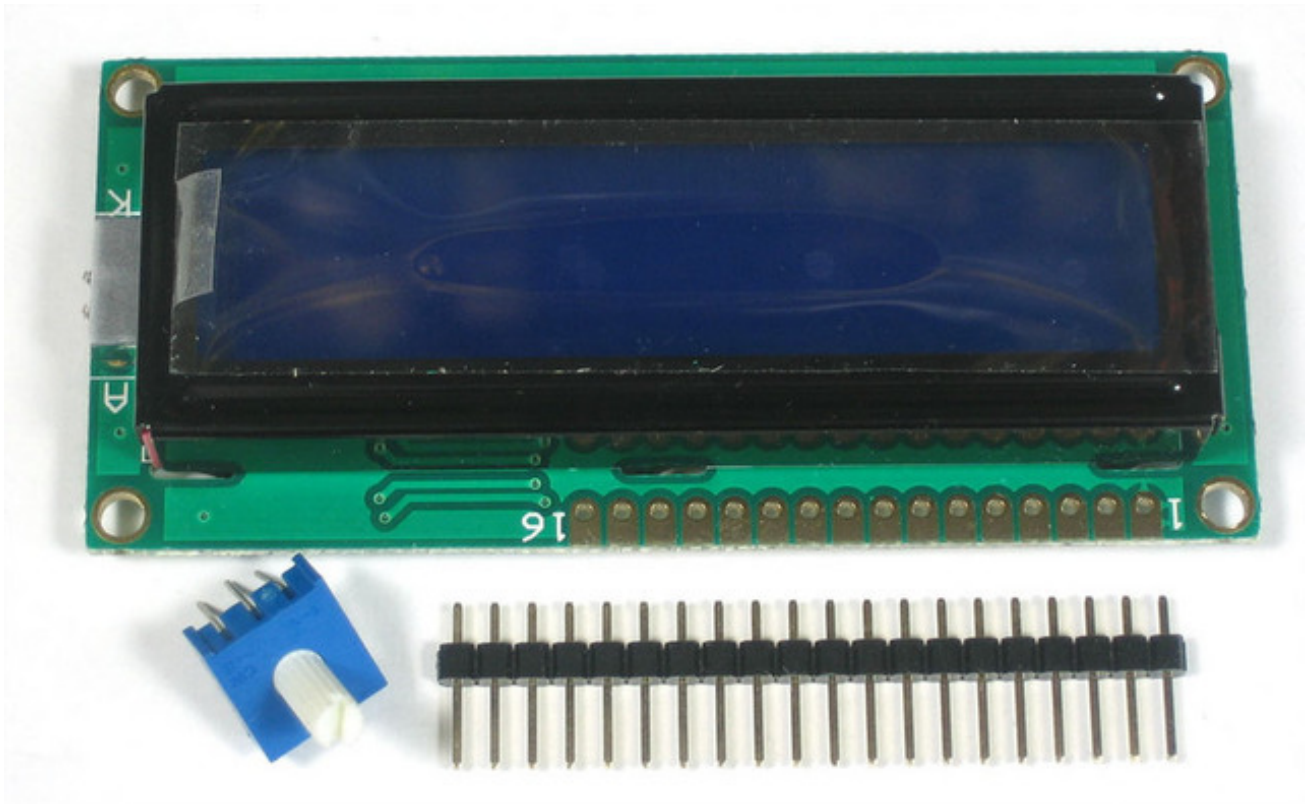
5v LCD vs 3.3v Pi

The raspberry Pi GPIOs are designed for 3.3v, but our LCD is a 5v device. **It's fine to use a 5v display, but only if we are sending data out of the Pi.** We are not going to use the 3.3v power rail on the cobbler and we will tie the **RW** (read/write) pin of the display to GND as we do not want the display sending a 5v signal into the Pi.

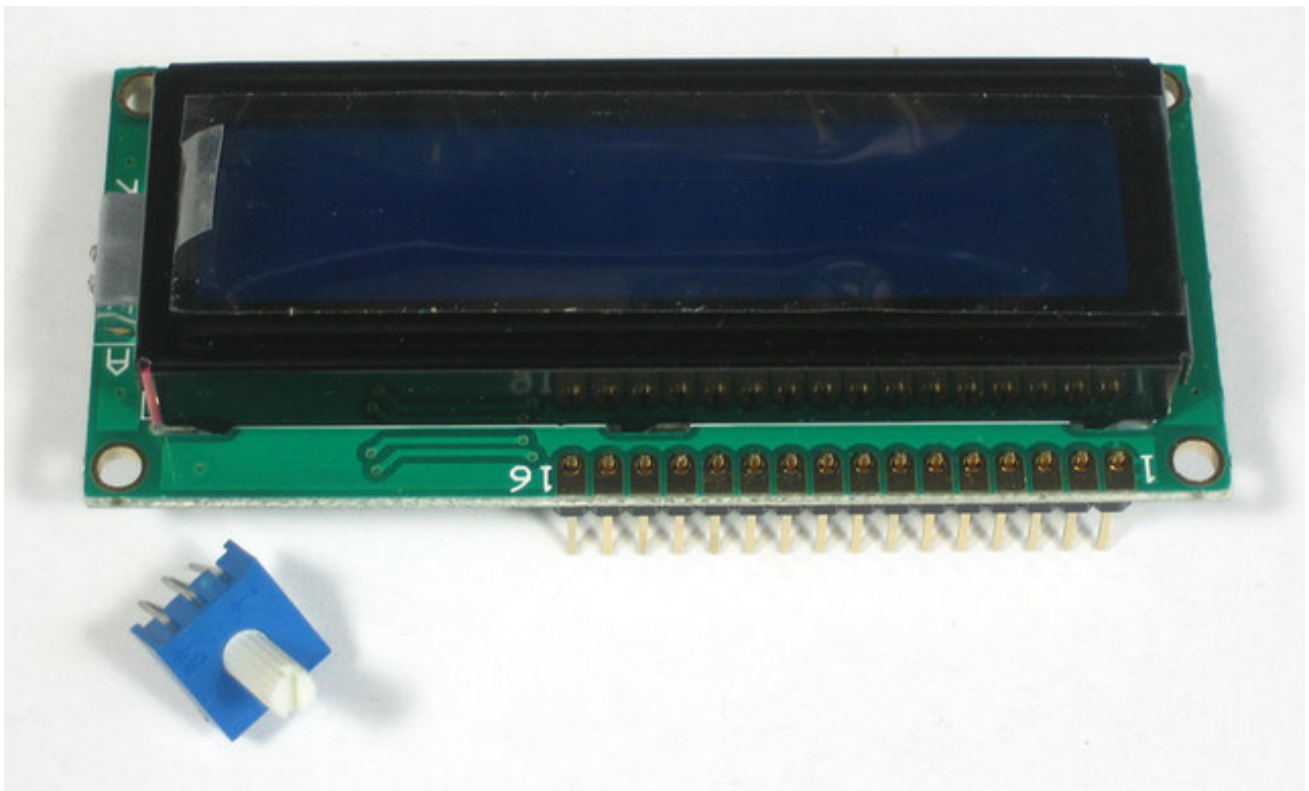
Don't cross the streams!

Preparing the LCD

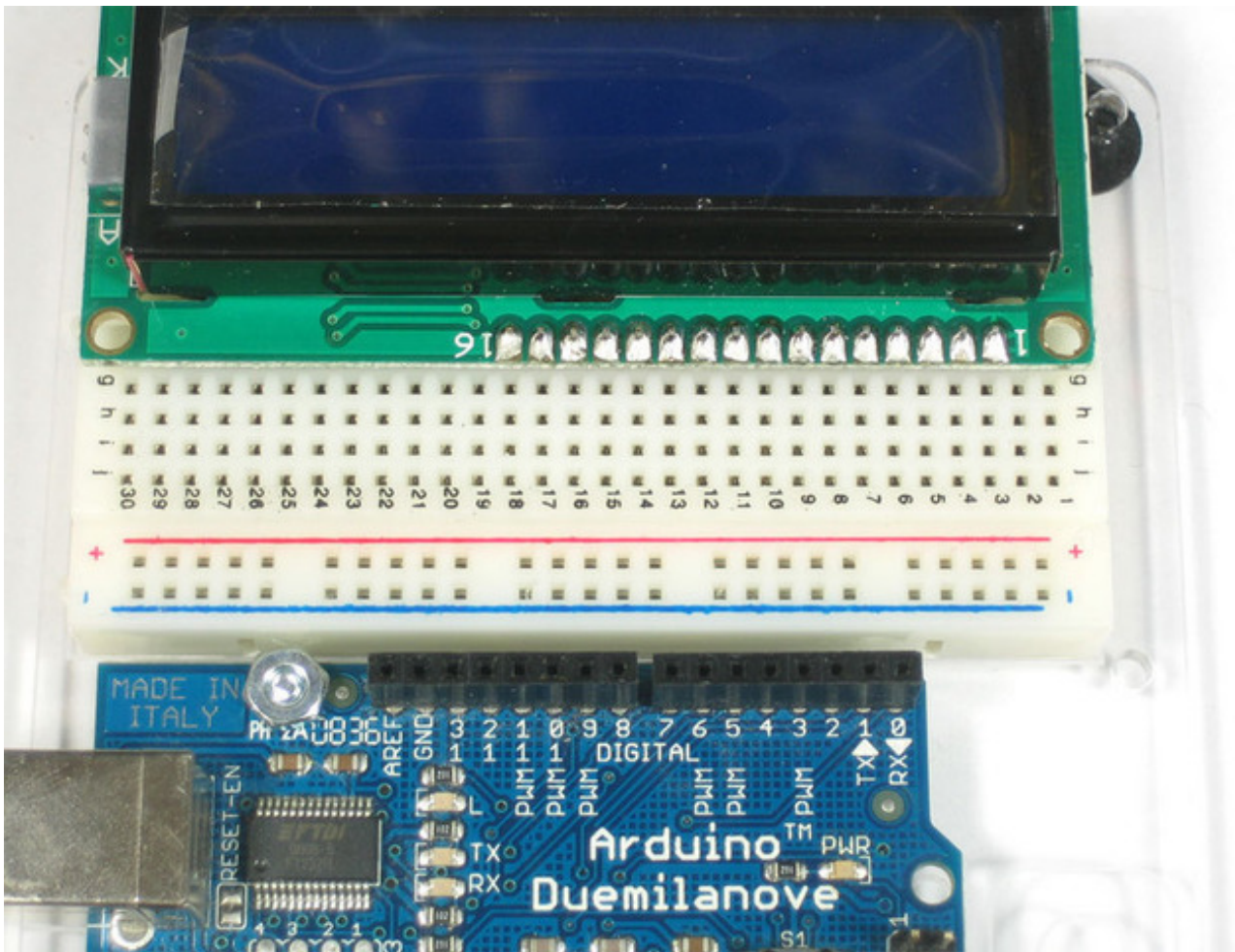
Before you start, make sure you have a strip of 0.1" male header and a 10K potentiometer. All Adafruit Character LCDs come with these parts so you should be good to go.



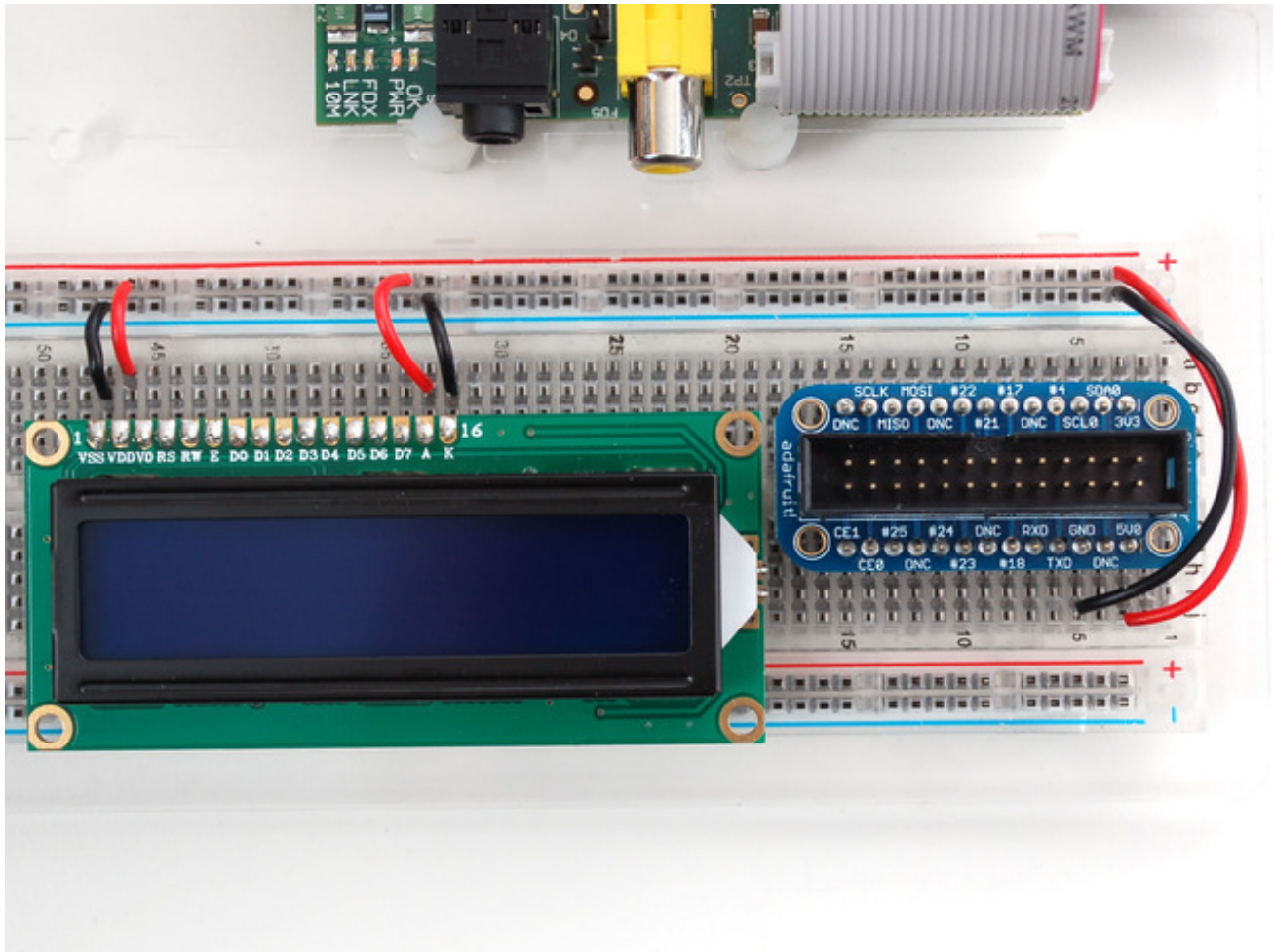
Most LCDs have a strip of 16 pins on the top, if the header is a little longer, just break it off until its the right length

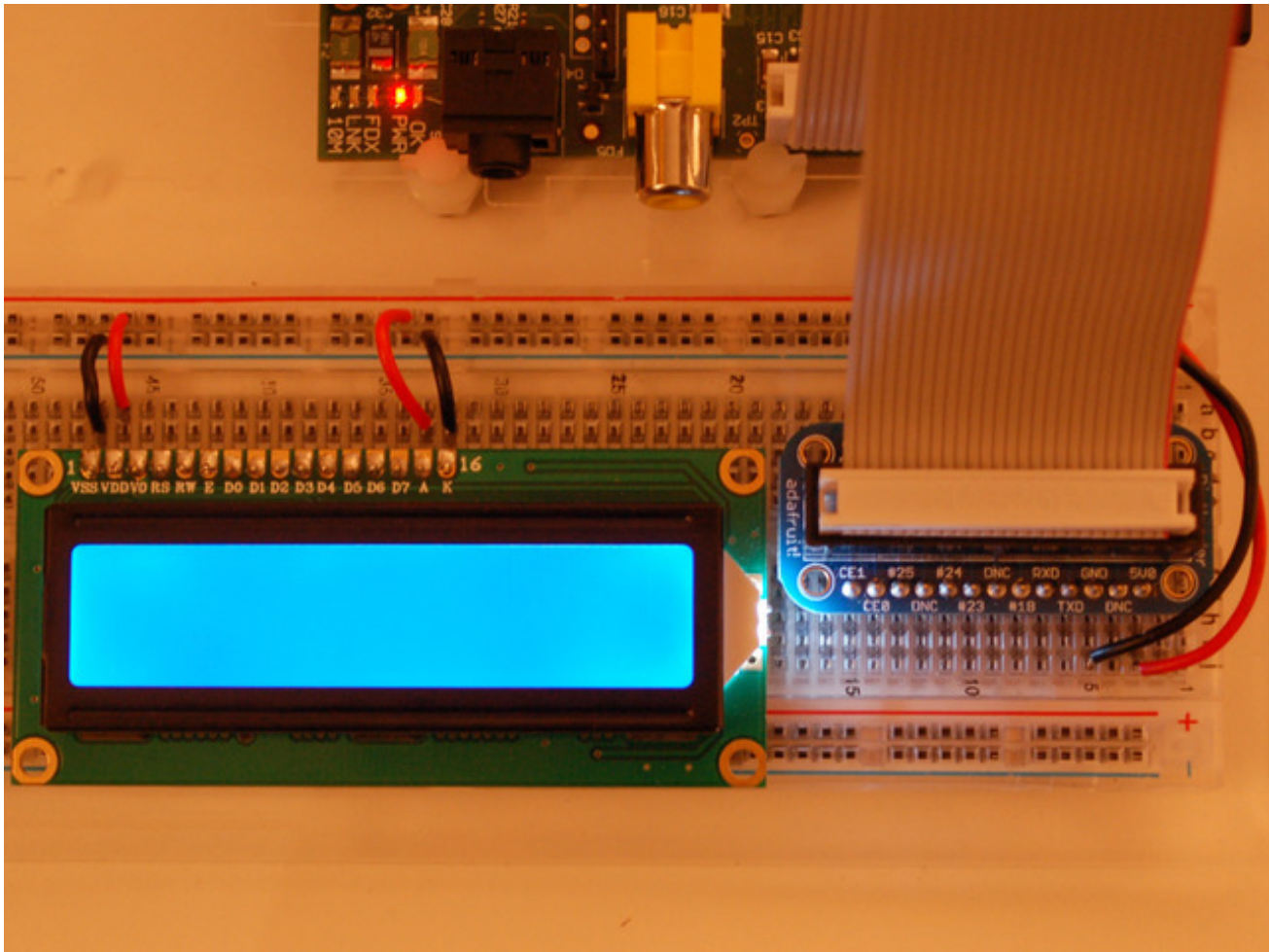


Next you'll need to solder the header to the LCD. **You must do this, it is not OK to just try to 'press fit' the LCD!**



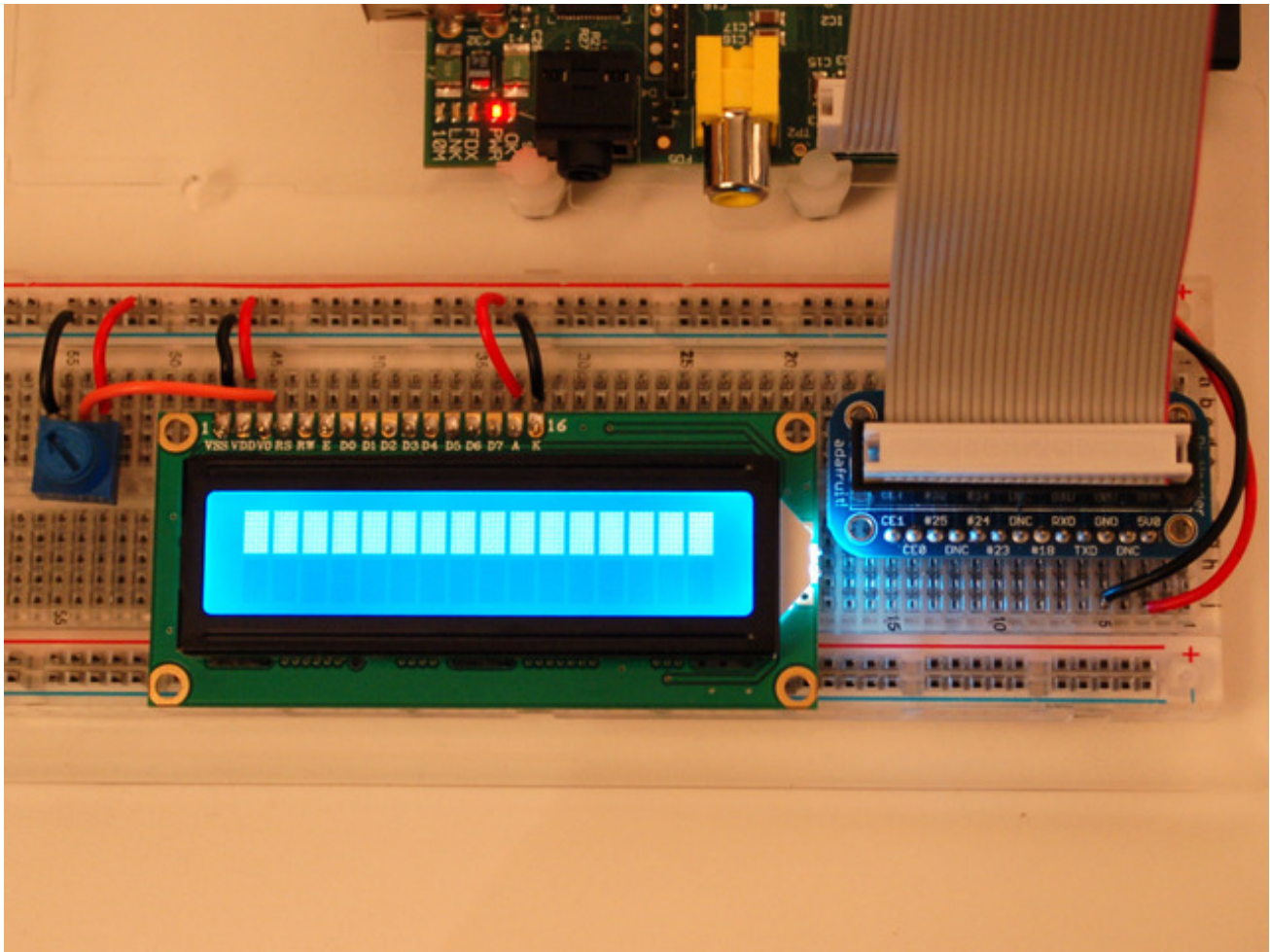
Start by connecting the **5V** and **GND** wires from the cobbler to the breadboard. Then connect pins #1, #2 and #15, #16 to the breadboard power rails as shown. The backlight should come on. If it doesn't, check the wiring!



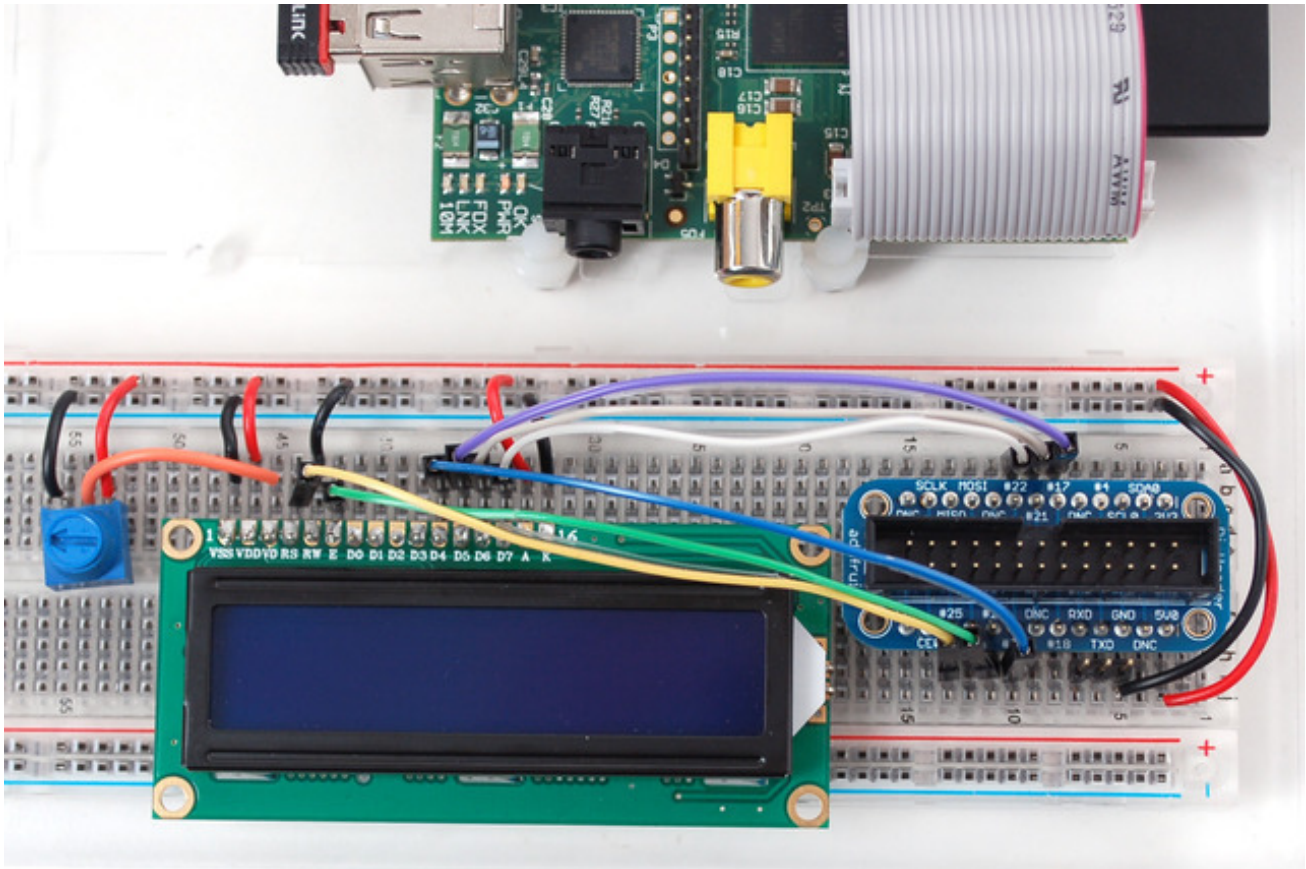


Next, wire up the contrast potentiometer as shown above, with the middle pin connecting to LCD pin #3 and the other two pins going to 5V and ground.

Twist the potentiometer until you see the first line of the LCD fill with boxes. If you don't see the boxes, check your wiring!



Finish the wiring for the **RS**, **RW**, **EN**, **D4**, **D5**, **D6**, and **D7** pins as shown in the diagram up top



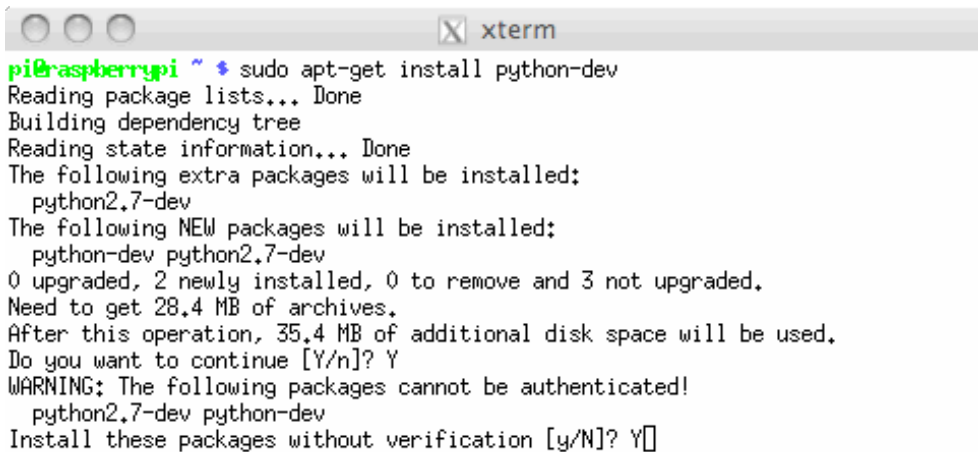
That's it! Now you're ready to run the python script to draw text on the screen!

Necessary Packages

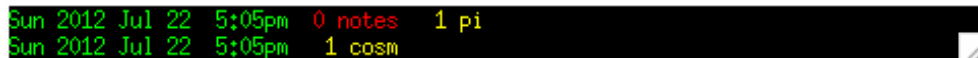
This guide is based on Debian's "Wheezy" release for Raspberry Pi. It was made available in Mid July 2012. The following items must be installed in order to utilize the Raspberry Pi's GPIO pins. If you are running [Adafruit's Occidentalis](http://adafru.it/aOp) (<http://adafru.it/aOp>) you can just skip this page.

Add the latest dev packages for Python (2.x)

```
$ sudo apt-get install python-dev
```



```
pi@raspberrypi ~$ sudo apt-get install python-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  python2.7-dev
The following NEW packages will be installed:
  python-dev python2.7-dev
0 upgraded, 2 newly installed, 0 to remove and 3 not upgraded.
Need to get 28.4 MB of archives.
After this operation, 35.4 MB of additional disk space will be used.
Do you want to continue [Y/n]? Y
WARNING: The following packages cannot be authenticated!
  python2.7-dev python-dev
Install these packages without verification [y/N]? Y
```



```
Sun 2012 Jul 22 5:05pm 0 notes 1 pi
Sun 2012 Jul 22 5:05pm 1 cosm
```

Upgrade distribute (required for RPi.GPIO 0.3.1a) - [No image for this one]

```
$ sudo apt-get install python-setuptools
$ sudo easy_install -U distribute
```

```
$ sudo apt-get install python-pip
```

```
xterm
pi@raspberrypi ~ $ sudo apt-get install python-pip
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  python-pkg-resources python-setuptools python2.6 python2.6-minimal
Suggested packages:
  python-distribute python-distribute-doc python2.6-doc binfmt-support
Recommended packages:
  python-dev-all
The following NEW packages will be installed:
  python-pip python-pkg-resources python-setuptools python2.6
  python2.6-minimal
0 upgraded, 5 newly installed, 0 to remove and 3 not upgraded.
Need to get 4,474 kB of archives.
After this operation, 14.5 MB of additional disk space will be used.
Do you want to continue [Y/n]?
WARNING: The following packages cannot be authenticated!
  python2.6-minimal python2.6 python-pkg-resources python-setuptools
  python-pip
Install these packages without verification [y/N]? Y
Get:1 http://mirrordirector.raspbian.org/raspbian/ wheezy/main python2.6-minimal
  armhf 2.6.8-0.2 [1,407 kB]
Sun 2012 Jul 22 4:59pm 1 cosm
```

Install rpi.gpio (0.3.1a) or later

```
$ sudo pip install rpi.gpio
```

```
xterm
Processing dependencies for distribute
Finished processing dependencies for distribute
pi@raspberrypi ~ $ sudo pip install rpi.gpio
Downloading/unpacking rpi.gpio
  Running setup.py egg_info for package rpi.gpio

Installing collected packages: rpi.gpio
  Running setup.py install for rpi.gpio
    building 'RPi.GPIO' extension
      gcc -pthread -fno-strict-aliasing -DDEBUG -g -fwrapv -O2 -Wall -Wstrict-pro
      totypes -fPIC -I/usr/include/python2.7 -c source/py_gpio.c -o build/temp.linu
      x-armv6l-2.7/source/py_gpio.o
      gcc -pthread -fno-strict-aliasing -DDEBUG -g -fwrapv -O2 -Wall -Wstrict-pro
      totypes -fPIC -I/usr/include/python2.7 -c source/c_gpio.c -o build/temp.linu
      x-armv6l-2.7/source/c_gpio.o
      gcc -pthread -shared -Wl,-O1 -Wl,-Bsymbolic-functions -Wl,-z,relro build/tem
      p.linux-armv6l-2.7/source/py_gpio.o build/temp.linux-armv6l-2.7/source/c_g
      pio.o -o build/lib.linux-armv6l-2.7/RPi.GPIO.so

Successfully installed rpi.gpio
Cleaning up...
pi@raspberrypi ~ $
Sun 2012 Jul 22 5:16pm 0 notes 1 pi
Sun 2012 Jul 22 5:16pm 1 cosm
```

Python Script

NOTE: This page describes an older version of the Raspberry Pi character LCD Python code. You can find a newer library that works on more platforms like the Raspberry Pi and BeagleBone Black by checking out the [new character LCD Python library guide \(http://adafru.it/dKa\)](http://adafru.it/dKa). For new projects it's recommended to use the new library!

The Code

The Python code for Adafruit's Character LCDs on the Pi is available on Github at <https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code> (<http://adafru.it/aOg>)

There are two files we will be working with:

1. **Adafruit_CharLCD.py**- contains python class for LCD control
2. **Adafruit_CharLCD_IPclock_example.py** - code for IP address and date/time calls methods from Adafruit_CharLCD.py

The first file *Adafruit_CharLCD.py* is a mashup from two different sources of LCD code. Github user lrvick put together a [nice python class \(http://adafru.it/aOq\)](http://adafru.it/aOq). His work is the baseline that is slowly being changed as we are bringing in more elements from the [Arduino LiquidCrystal library \(http://adafru.it/aOr\)](http://adafru.it/aOr).

The easiest way to get the code onto your Pi is to hook up an Ethernet cable, and clone it directly using 'git', which is installed by default on most distros. Simply run the following commands from an appropriate location (ex. "/home/pi"):

```
apt-get install git
git clone git://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code.git
cd Adafruit-Raspberry-Pi-Python-Code
cd Adafruit_CharLCD
```

Testing

You can test the wiring from the previous step by simply running the *Adafruit_CharLCD.py* python code, as it has a little code in that will simply display a test message when wired correctly

If you're using a Version 2 Raspberry Pi, pin **#21** has been replaced with pin **#27** so edit **Adafruit_CharLCD.py** and change:

```
def __init__(self, pin_rs=25, pin_e=24, pins_db=[23, 17, 21, 22], GPIO = None):
```

to

```
def __init__(self, pin_rs=25, pin_e=24, pins_db=[23, 17, 27, 22], GPIO = None):
```

you can use **nano Adafruit_CharLCD.py** to edit

```
chmod +x Adafruit_CharLCD.py
sudo ./Adafruit_CharLCD.py
```

IP Clock Example

This script assumes you'll want to display the Ethernet (eth0) IP address. You may want to replace eth0 with wlan0 or wlan1 (etc) for Wireless IP address!

```
#!/usr/bin/python

from Adafruit_CharLCD import Adafruit_CharLCD
from subprocess import *
from time import sleep, strftime
from datetime import datetime

lcd = Adafruit_CharLCD()

cmd = "ip addr show eth0 | grep inet | awk '{print $2}' | cut -d/ -f1"

lcd.begin(16,1)

def run_cmd(cmd):
    p = Popen(cmd, shell=True, stdout=PIPE)
    output = p.communicate()[0]
    return output

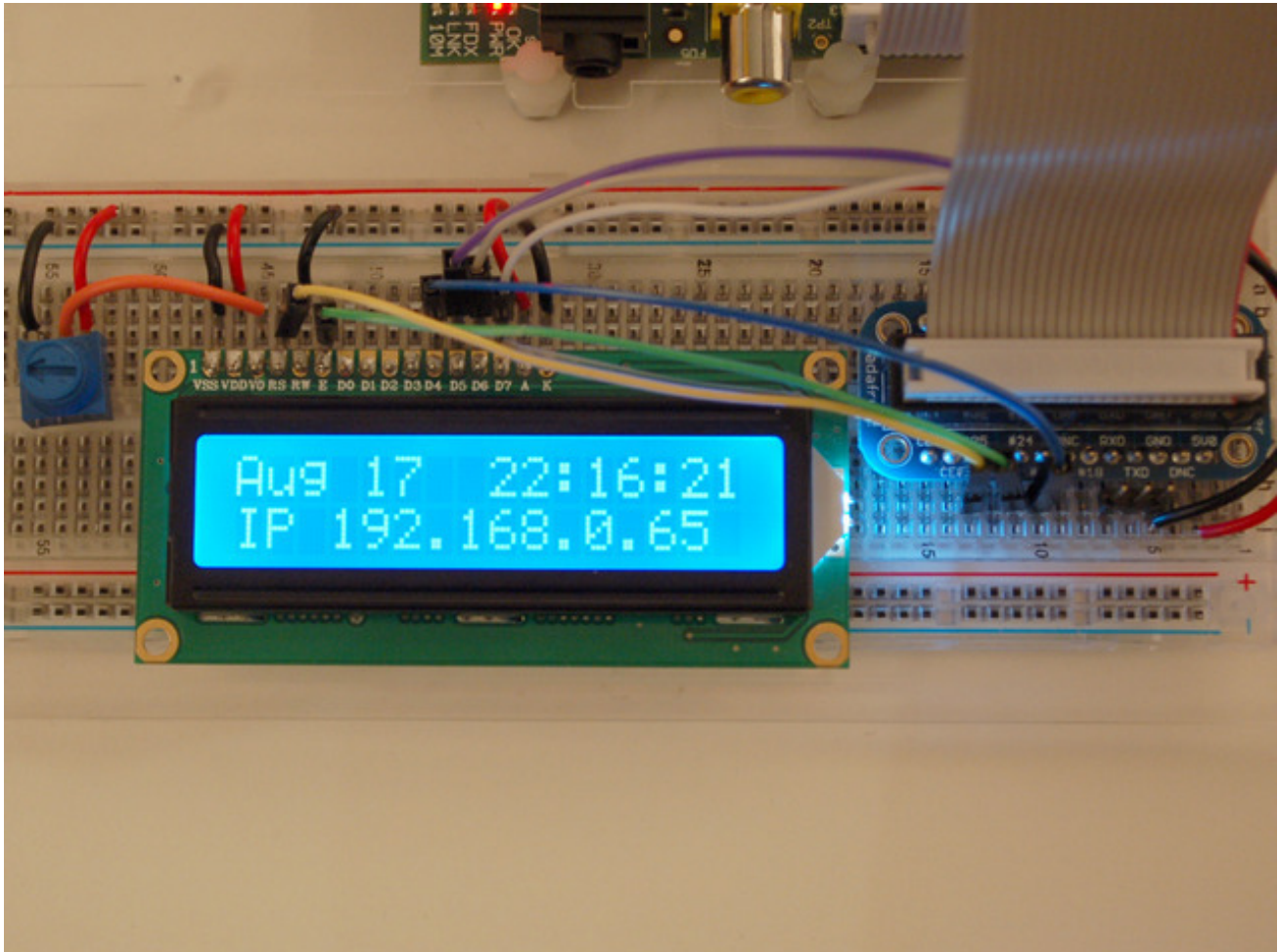
while 1:
    lcd.clear()
    ipaddr = run_cmd(cmd)
    lcd.message(datetime.now().strftime('%b %d %H:%M:%S\n'))
    lcd.message('IP %s' % ( ipaddr ) )
    sleep(2)
```

Running the Code

Start the IP + Clock example

```
$ sudo ./Adafruit_CharLCD_IPclock_example.py
```

What You Should See



Init Script

It's all fine and dandy to have a script like *Adafruit_CharLCD_IPclock_example.py* which we can manually run, but wouldn't it be nice to have the time and ip address pop up on the display when the raspberry pi boots up? This is done using a init script which runs the *Adafruit_CharLCD_IPclock_example.py* code on boot and kills it during system shut down.

Paste this code into **/etc/init.d/lcd**
(you will need to use sudo to write to this directory)

```
### BEGIN INIT INFO
# Provides: LCD - date / time / ip address
# Required-Start: $remote_fs $syslog
# Required-Stop: $remote_fs $syslog
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Liquid Crystal Display
# Description: date / time / ip address
### END INIT INFO

#!/bin/sh

# /etc/init.d/lcd

export HOME
case "$1" in
    start)
        echo "Starting LCD"
        /home/pi/Adafruit-Raspberry-Pi-Python-Code/Adafruit_CharLCD/Adafruit_CharLCD_IPclock_exam
        ;;
    stop)
        echo "Stopping LCD"
        LCD_PID=`ps auxwww | grep Adafruit_CharLCD_IPclock_example.py | head -1 | awk '{print $2}'`
        kill -9 $LCD_PID
        ;;
    *)
        echo "Usage: /etc/init.d/lcd {start|stop}"
        exit 1
        ;;
esac
exit 0
```

You should change

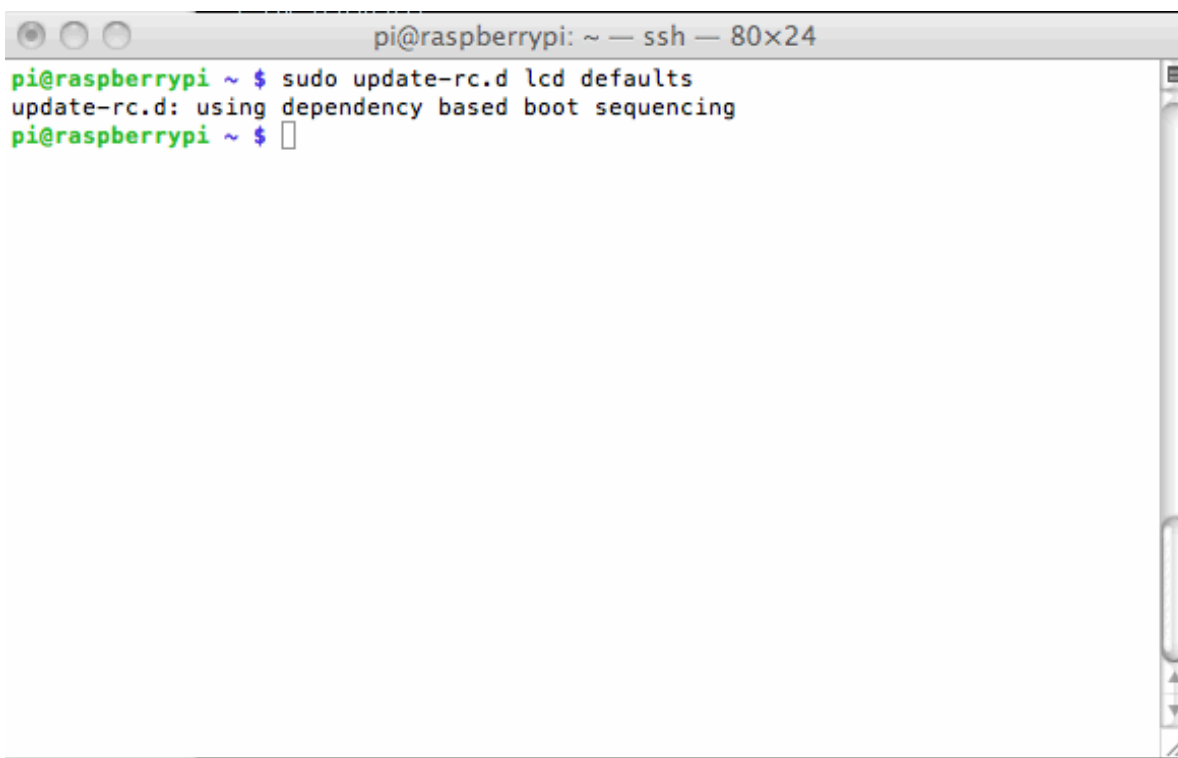
/home/pi/Adafruit-Raspberry-Pi-Python-Code/Adafruit_CharLCD/Adafruit_CharLCD_IPclock_example.py to where-ever you are actually keeping the IPclock python script

Make the init script executable.

```
$ sudo chmod +x /etc/init.d/lcd
```

Make the lcd init script known to the system by using the update-rc.d command.

```
$ sudo update-rc.d lcd defaults
```

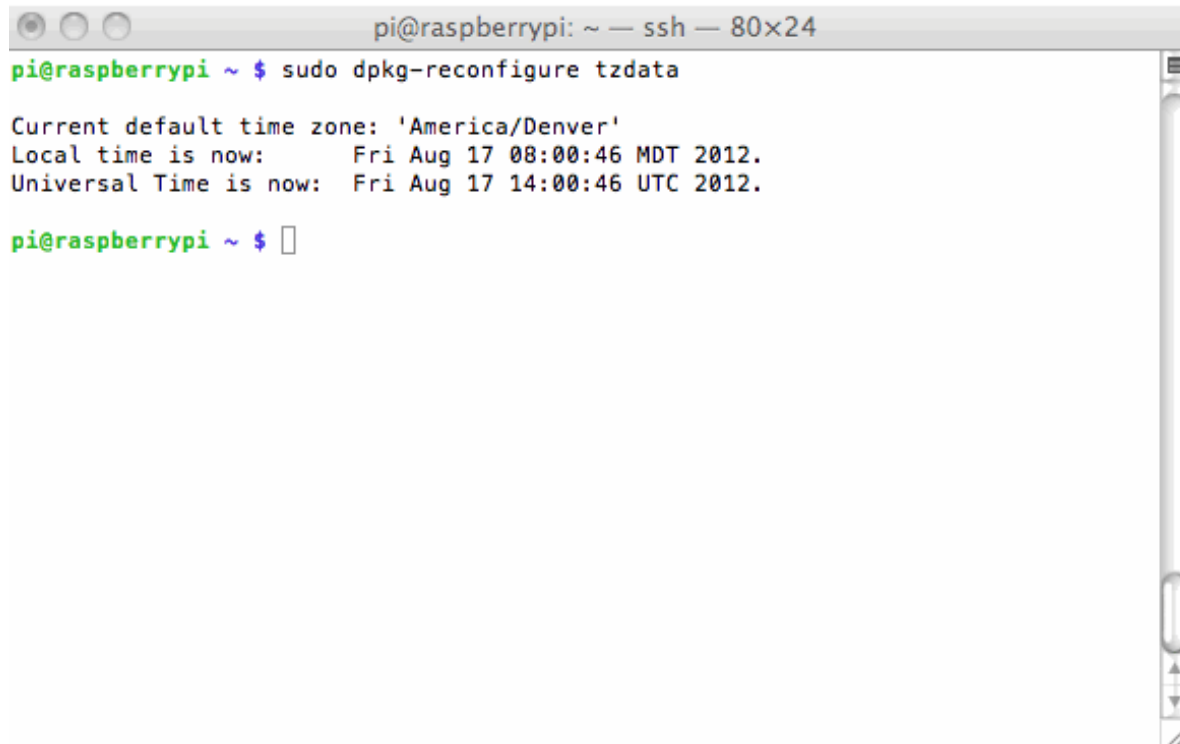
A screenshot of a terminal window titled 'pi@raspberrypi: ~ — ssh — 80x24'. The terminal shows the command 'sudo update-rc.d lcd defaults' being entered and executed. The output of the command is 'update-rc.d: using dependency based boot sequencing'. The prompt returns to 'pi@raspberrypi ~ \$'.

Now on each boot the lcd will automatically show the date/time/ip address on startup. This means you will know when the pi is reachable and what the ip address is without having to plug a monitor in.

Time Zone

Last, but not least. My pi came configured with UT (Universal Time). I prefer to see time based on my time zone which is Mountain. Here is how to configure time on the pi for any location. This is a one time configuration setting that will survive between reboots.

```
$ sudo dpkg-reconfigure tzdata
```



```
pi@raspberrypi ~ $ sudo dpkg-reconfigure tzdata

Current default time zone: 'America/Denver'
Local time is now:      Fri Aug 17 08:00:46 MDT 2012.
Universal Time is now:  Fri Aug 17 14:00:46 UTC 2012.

pi@raspberrypi ~ $
```

The command launches a curses based program which allows arrow keys to be used to select the region specific time zone.

