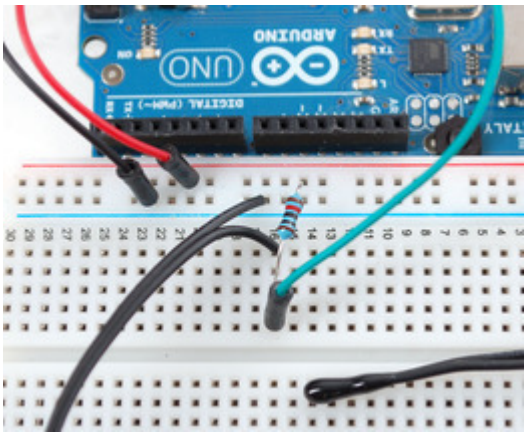
Using a Thermistor

Connecting to a Thermistor

These thermistors are pretty hardy, you can strip the PVC insulation and stick the wires into a breadboard or solder to them directly. Of course you can cut or extend the wires. Since the resistance is pretty high (10Kohm) the wire resistance won't make a huge difference.



Analog Voltage Reading Method

To measure the temperature, we need to measure the resistance. However, a microcontroller does not have a resistance-meter built in. Instead, it only has a voltage reader known as a analog-digital-converter. So what we have to do is convert the resistance into a voltage, and we'll do that by adding another resistor and connecting them in series. Now you just measure the voltage in the middle, as the resistance changes, the voltage changes too, according to the simple voltage-divider equation. We just need to keep one resistor fixed

Say the fixed resistor is **10K** and the variable resistor is called **R** - the voltage output (**Vo**) is:

Vo = R / (R + 10K) * Vcc

Where **Vcc** is the power supply voltage (3.3V or 5V)

Now we want to connect it up to a microcontroller. Remember that when you measure a voltage (**Vi**) into an Arduino ADC, you'll get a number.

ADC value = Vi * 1023 / Varef

So now we combine the two (**Vo** = **Vi**) and get:

ADC value = **R / (R + 10K) * Vcc * 1023 / Varef**

What is nice is that if you notice, if Vcc (logic voltage) is the same as the ARef, analog reference voltage, the values cancel out!

ADC value = **R / (R + 10K) * 1023**

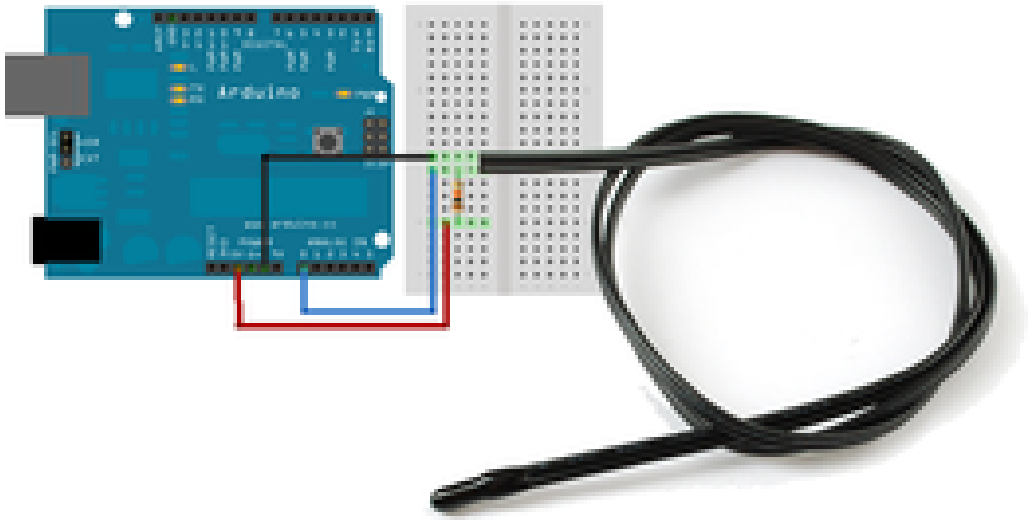
It doesn't matter what voltage you're running under. Handy!

Finally, what we really want to do is get that **R** (the unknown resistance). So we do a little math to move the **R** to one side:

R = 10K / (1023/ADC - 1)

*Lots of people have emailed me to tell me the above equation is wrong and the correct calculation is **R = 10K*ADC / (1023 - ADC)**. Their equivalence is left as an exercise for the reader! ;)*

Great, lets try it out. Connect up the thermistor as shown:



Connect one end of the 10K resistor to 5V, connect the other end of the 10K 1% resistor to one pin of the thermistor and the other pin of the thermistor to ground. Then connect Analog 0 pin to the 'center' of the two.

Now run the following sketch:

Download Project Bundle
Copy Code

```
// thermistor-1.ino Simple test program for a thermistor for Adafruit Learning
System
// https://learn.adafruit.com/thermistor/using-a-thermistor by Limor Fried,
Adafruit Industries
// MIT License - please keep attribution and consider buying parts from Adafruit

// the value of the 'other' resistor
#define SERIESRESISTOR 10000

// What pin to connect the sensor to
#define THERMISTORPIN A0

void setup(void) {
  Serial.begin(9600);
}

void loop(void) {
  float reading;

  reading = analogRead(THERMISTORPIN);

  Serial.print("Analog reading ");
  Serial.println(reading);

  // convert the value to resistance
  reading = (1023 / reading)  - 1;     // (1023/ADC - 1)
  reading = SERIESRESISTOR / reading;  // 10K / (1023/ADC - 1)
  Serial.print("Thermistor resistance ");
  Serial.println(reading);

  delay(1000);
}
```

View on GitHub
You should get responses that correspond to the resistance of the thermistor as measured with a multimeter

If you are not getting correct readings, check that the 10K resistor is placed between VCC and A0, and the thermistor is between A0 and ground. Check you have a 10K Thermistor and that you are using a 'standard' NTC thermistor. On a "5V" microcontroller like classic Arduino or Metro 328, use 5V for the VCC pin. On 3.3V microcontrollers like Feather or Arduino Zero, use 3.3V for the VCC pin.

If, when you heat up the thermistor, the temperature reading goes down, check that you don't have the two resistors swapped and check that you are using an NTC not PTC thermistor.

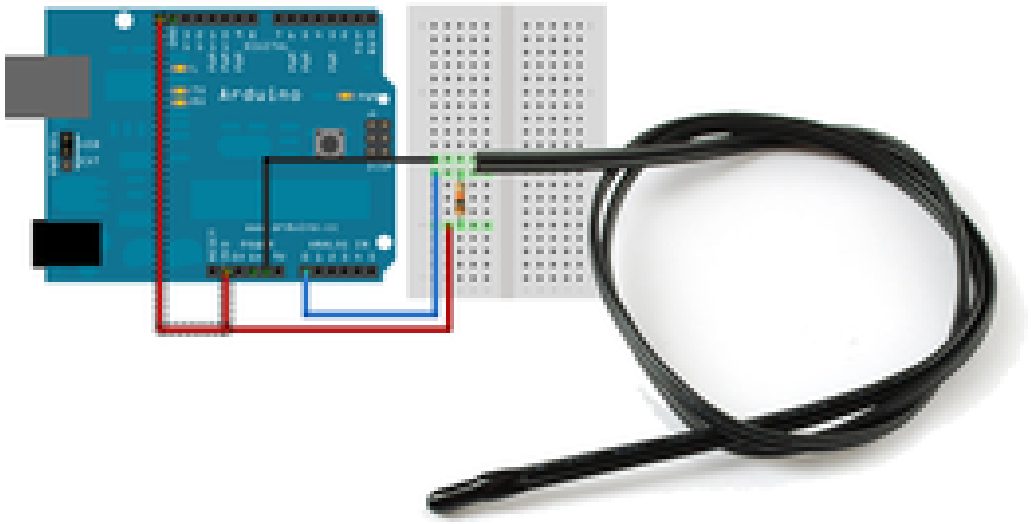
Better Readings

When doing analog readings, especially with a 'noisy' board like the arduino, we suggest two tricks to improve results. One is to use the 3.3V voltage pin as an analog reference and the other is to take a bunch of readings in a row and average them.

The first trick relies on the fact that the 5V power supply that comes straight from your computer's USB does a lot of stuff on the Arduino, and is almost always much noisier than the 3.3V line (which goes through a secondary filter/regulator stage!) It's easy to use, simply connect 3.3V to AREF and use that as the VCC voltage. Because our calcuations don't include the VCC voltage, you don't have to change your equation. You do have to set the analog reference but that's a single line of code

Taking multiple readings to average out the result helps get slightly better results as well, since you may have noise or fluctuations, we suggest about 5 samples.

Rewire as shown, the 10K resistor is still connected to the higher voltage, and the thermistor to ground



This sketch takes those two improvements and integrates them into the demo, you will have better, more precise readings.

Note that this code specifies an EXTERNAL voltage reference. To work properly, you must make the additional connection to the AREF pin as shown in the diagram above.

Download Project Bundle
Copy Code

```
// thermistor-2.ino Intermediate test program for a thermistor. Adafruit Learning
System Tutorial
// https://learn.adafruit.com/thermistor/using-a-thermistor by Limor Fried,
Adafruit Industries
// MIT License - please keep attribution and please consider buying parts from
Adafruit

// which analog pin to connect
#define THERMISTORPIN A0
// how many samples to take and average, more takes longer
// but is more 'smooth'
#define NUMSAMPLES 5
// the value of the 'other' resistor
#define SERIESRESISTOR 10000

int samples[NUMSAMPLES];

void setup(void) {
  Serial.begin(9600);
  // connect AREF to 3.3V and use that as VCC, less noisy!
  analogReference(EXTERNAL);
}

void loop(void) {
  uint8_t i;
  float average;

  // take N samples in a row, with a slight delay
  for (i=0; i< NUMSAMPLES; i++) {
   samples[i] = analogRead(THERMISTORPIN);
   delay(10);
  }

  // average all the samples out
  average = 0;
  for (i=0; i< NUMSAMPLES; i++) {
     average += samples[i];
  }
  average /= NUMSAMPLES;

  Serial.print("Average analog reading ");
  Serial.println(average);
  // convert the value to resistance
  average = 1023 / average - 1;
  average = SERIESRESISTOR / average;

  Serial.print("Thermistor resistance ");
  Serial.println(average);

  delay(1000);
}
```

View on GitHub

Converting to Temperature

Finally, of course, we want to have the temperature reading, not just a resistance! If you just need to do a quick comparison circuit (if temperature is below X do this, if its above Y do that), you can simply use the temperature/resistance table which correlates the resistance of the thermistor to the temperature.

However, you probably want actual temperature values. To do that we'll use the Steinhart-Hart equation , which lets us do a good approximation of converting values. Its not as exact as the thermistor table (it is an approximation) but its pretty good around the temperatures that this thermistor is used.

However, this equation is fairly complex, and requires knowing a lot of variables

$$\frac{1}{T} = A + B\ln(R) + C(\ln(R))^3$$

that we don't have for this thermistor. Instead we will use the simplified B parameter equation.

For this one we only need to know **To** (which is room temperature, 25 °C = 298.15 K) **B** (in this case 3950, the coefficient of the thermistor), and **Ro** (the resistance at room temp, in this case 10Kohm). We plug in **R** (resistance measured) and get out **T** (temperature in Kelvin) which is easy to convert to °C

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{B}\ln\left(\frac{R}{R_0}\right)$$

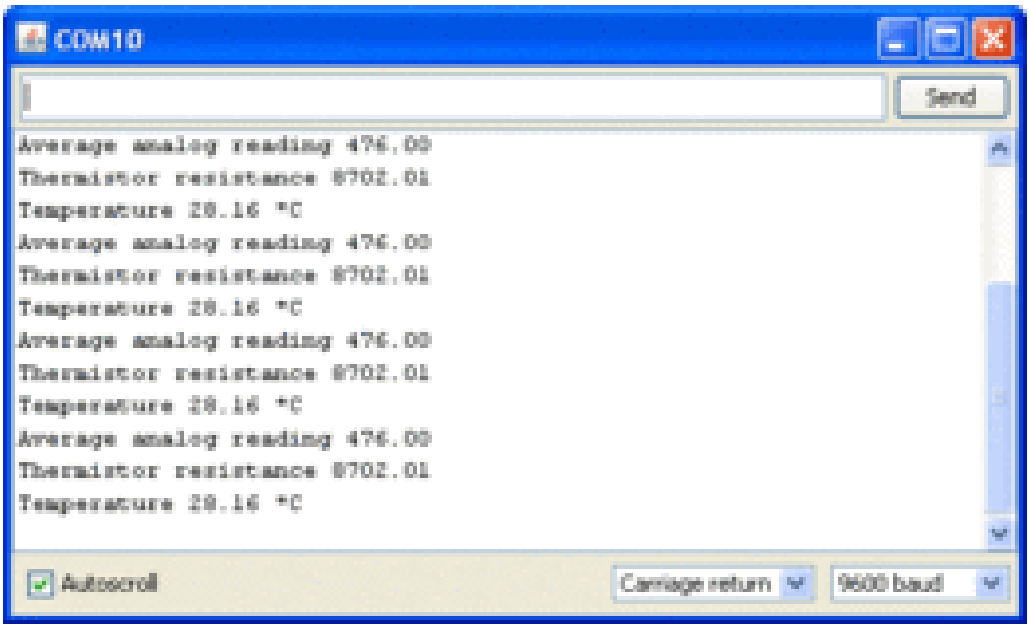
The following sketch will calculate °C for you

```
// Thermistor Example #3 from the Adafruit Learning System guide on Thermistors
// https://learn.adafruit.com/thermistor/overview by Limor Fried, Adafruit
Industries
// MIT License - please keep attribution and consider buying parts from Adafruit

// which analog pin to connect
#define THERMISTORPIN A0
// resistance at 25 degrees C
#define THERMISTORNOMINAL 10000
// temp. for nominal resistance (almost always 25 C)
#define TEMPERATURENOMINAL 25
// how many samples to take and average, more takes longer
// but is more 'smooth'
#define NUMSAMPLES 5
// The beta coefficient of the thermistor (usually 3000-4000)
#define BCOEFFICIENT 3950
// the value of the 'other' resistor
#define SERIESRESISTOR 10000

int samples[NUMSAMPLES];

void setup(void) {
  Serial.begin(9600);
  analogReference(EXTERNAL);
}

void loop(void) {
  uint8_t i;
  float average;

  // take N samples in a row, with a slight delay
  for (i=0; i< NUMSAMPLES; i++) {
   samples[i] = analogRead(THERMISTORPIN);
   delay(10);
  }

  // average all the samples out
  average = 0;
  for (i=0; i< NUMSAMPLES; i++) {
     average += samples[i];
  }
  average /= NUMSAMPLES;

  Serial.print("Average analog reading ");
  Serial.println(average);

  // convert the value to resistance
  average = 1023 / average - 1;
  average = SERIESRESISTOR / average;
  Serial.print("Thermistor resistance ");
  Serial.println(average);

  float steinhart;
  steinhart = average / THERMISTORNOMINAL;     // (R/Ro)
  steinhart = log(steinhart);                  // ln(R/Ro)
  steinhart /= BCOEFFICIENT;                   // 1/B * ln(R/Ro)
  steinhart += 1.0 / (TEMPERATURENOMINAL + 273.15); // + (1/To)
  steinhart = 1.0 / steinhart;                 // Invert
  steinhart -= 273.15;                         // convert absolute temp to C
```

```
  Serial.print("Temperature ");
  Serial.print(steinhart);
  Serial.println(" *C");

  delay(1000);
}
```

View on GitHub

For better precision, we suggest reading the exact value of the 'series 10K' it should be nearly exactly 10K but if you can get a better reading that will reduce your error.



How Accurate is the Reading?

You may notice that above, the temperature reading is 28.16°C - does that mean we have 0.01°C accuracy? Unfortunately no! The thermistor has error and the analog reading circuitry has error.

We can approximate the expected error by first taking into account the thermistor resistance error. The thermistor is correct to 1%, which means that at 25°C it can read 10,100 to 9900 ohms. At around 25°C a difference of 450 ohms represents 1°C so 1% error means about +-0.25°C (you may be able to calibrate this away by determining the resistance of the thermistor in a 0°C ice bath and removing any offset). You can also spring for a 0.1% thermistor which will reduce the possible resistance error down to +-0.03°C

Then there is the error of the ADC, for every bit that it is wrong the resistance (around 25°C) can be off by about 50 ohms. This isn't too bad, and is a smaller error than the thermistor error itself +-(0.1°C) but there is no way to calibrate it 'away' - a higher precision ADC (12-16 bits instead of 10) will give you more precise readings

In general, we think thermistors are higher precision than thermocouples, or most low cost digital sensors, but you will not get better than +-0.1°C accuracy on an Arduino with a 1% thermistor and we would suggest assuming no better than +-0.5°C.

Self-Heating

If you have a 10K thermistor + 10K resistor connected between 5V and ground, you'll get about 5V / (10K + 10K) = 0.25mA flowing at all times. While this isn't a lot of current, it will heat up your thermistor as the 10K thermistor will be dissipating about 0.25mA * 2.5V = 0.625 mW.

To avoid this heating, you can try connecting the 'top' of the resistor divider to a GPIO pin and set that pin HIGH when you want to read (thus creating the divider) and then LOW when you are in low power mode (no current will flow from 0V to ground)