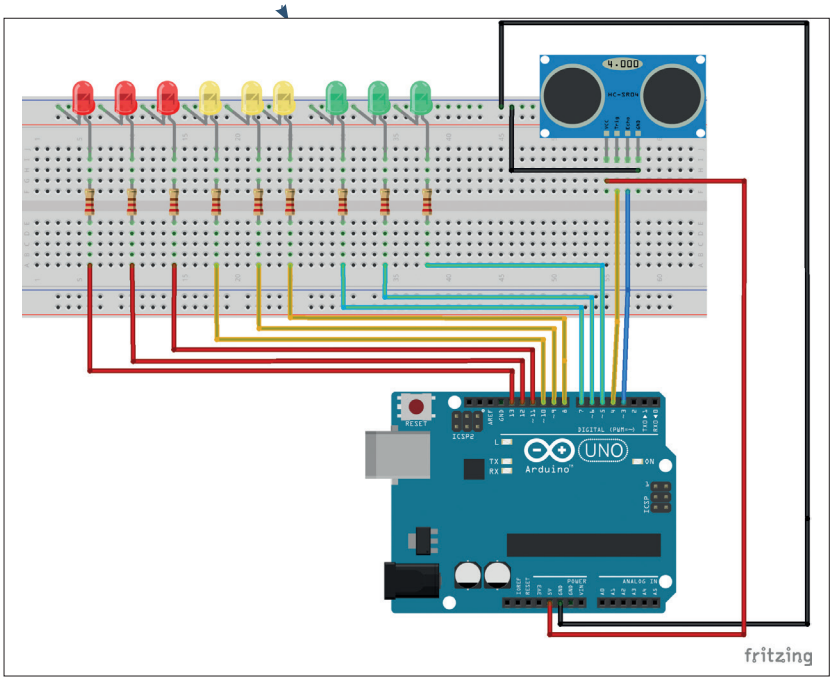


This diagram was created in *Fritzing*, a great tool to help you design the layout of a project. You can find a high resolution version in the repository for this project.

where they are connected and what they will do. This is achieved via the **void setup()** configuration section, and in here we use **pinMode** to instruct the Arduino on what each pin will do. We earlier created a series of variables that store the pin locations for each of the components. We will use those with **pinMode** to identify the pin that we wish to configure. The configuration is quite simple: is the pin an input or an output? An input will wait for a signal/current from an external component, while an output will send a signal/current to an external component.

```
void setup() {
 pinMode(trigPin, OUTPUT);
 pinMode(echoPin, INPUT);
 pinMode(red1, OUTPUT);
 pinMode(red2, OUTPUT);
 pinMode(red3, OUTPUT);
 pinMode(yel1, OUTPUT);
 pinMode(yel2, OUTPUT);
 pinMode(yel3, OUTPUT);
 pinMode(gre1, OUTPUT);
 pinMode(gre2, OUTPUT);
 pinMode(gre3, OUTPUT);
}
```

The **void loop()** is the our main body of code and contains the logic that controls the detection of an object by the ultrasonic sensor. We start the first part of this section by creating two variables, called **duration** and **distance**; these will contain long integers, so called because they have an extended size to store large numbers. We will use these variables to store the time taken for the pulse to be sent and received, and we shall use **distance** to store the answer to a calculation later in the code.

We next trigger a pulse to be sent from the ultrasonic sensor, but before we do that we must ensure that the ultrasonic sensor is not already transmitting. We do that using **digitalWrite**, which instructs the **trigPin** to change its state from on to off (**HIGH** to **LOW**).

The code then instructs the project to wait for two microseconds, which is just enough time for the ultrasonic sensor to settle ready for use.

We now use the **digitalWrite** function to send a pulse from the sensor by setting the **trigPin** to **HIGH**, in other words sending current to the sensor. Current is sent to the ultrasonic sensor for 10 microseconds using the **delayMicroseconds()** function. We then turn off the current to the **trigPin**, ending the pulse transmission sequence.

Now we need to do a little maths. To kick things off we record the time taken for the pulse to be sent and received, and this is stored in the **duration** variable that we created earlier. Lastly we use the **distance** variable to store the answer to the calculation, **duration** divided by 2, as we only need to know how long it took for the pulse to be received. The answer is then divided by 29.1 to give us the distance in centimetres:

```
void loop() {
 long duration, distance;
 digitalWrite(trigPin, LOW);
 delayMicroseconds(2);
 digitalWrite(trigPin, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPin, LOW);
 duration = pulseIn(echoPin, HIGH);
 distance = (duration/2) / 29.1;
```

For our last section of code we use the classic **if**, **else if**, **else** conditional statement to check for three different states. We'll start with the **if** statement.

The first condition that we wish to test is to check the distance between the sensor and any objects that might be in the way. At this time we're looking for objects less than 10 centimetres away, and if this condition is true we turn on the power to all of the red LEDs, and turn off the power to the yellow and green LED. This tells us that the object is really close, just like a parking sensor does in our cars:

```
if (distance < 10) {
  digitalWrite(red1,HIGH);
  digitalWrite(red2,HIGH);
  digitalWrite(red3,HIGH);
  digitalWrite(gre1,LOW);
  digitalWrite(gre2,LOW);
  digitalWrite(gre3,LOW);
  digitalWrite(yel1,LOW);
  digitalWrite(yel2,LOW);
  digitalWrite(yel3,LOW);
}
```

Our next condition to check uses an **else if** statement, and this means that if the first **if** statement is false, check to see if this **else if** statement is now true and if so run the code. So if the distance between our sensor and object is greater than 10 centimetres but less than 30 centimetres, the red and green LEDs are turned off, and the yellow LED are turned on, indicating that we are getting closer to the object:

```
else if (distance > 10 and distance < 30) {
  digitalWrite(yel1,HIGH);
  digitalWrite(yel2,HIGH);
  digitalWrite(yel3,HIGH);
  digitalWrite(gre1,LOW);
  digitalWrite(gre2,LOW);
  digitalWrite(gre3,LOW);
  digitalWrite(red1,LOW);
  digitalWrite(red2,LOW);
  digitalWrite(red3,LOW);
}
```

Our last condition to test is rather simple, as it does not require anything to test. **else** is used when all other conditions have been tested and proven to be false. If everything is false then **else** must be true. So if the object is not less than 10 centimetres away, or further than 30 centimetres away then the red and yellow LED will be turned off and the green LED will be turned on, indicating that we are far enough away from the sensor. Our last line of code controls the speed of the project and introduces a half-second delay before the main loop is repeated once again:

```
else {
  digitalWrite(red1,LOW);
  digitalWrite(red2,LOW);
  digitalWrite(red3,LOW);
  digitalWrite(gre1,HIGH);
  digitalWrite(gre2,HIGH);
  digitalWrite(gre3,HIGH);
  digitalWrite(yel1,LOW);
  digitalWrite(yel2,LOW);
  digitalWrite(yel3,LOW);
 }
 delay(500);
}
```

Building the hardware

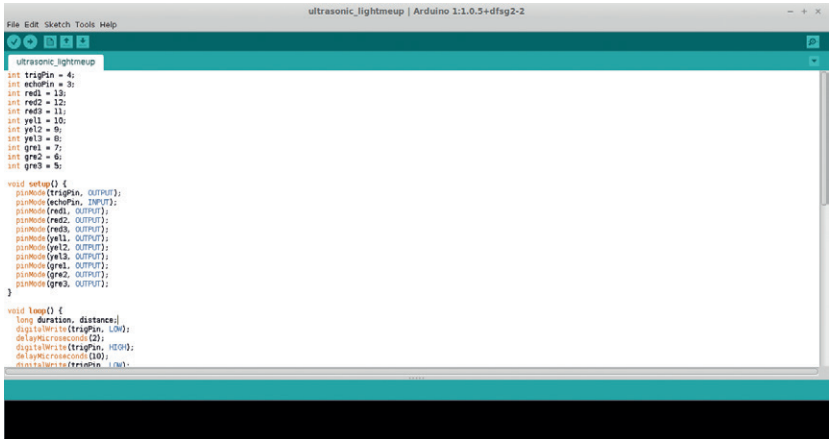
Arduino projects come as a package, with software and hardware. With the code already taken care of, our focus shifts to the hardware build of the project.

We start our build with the humble breadboard, and to the breadboard we add the HC-SR04 ultrasonic sensor, taking care to note the pin layout as we will need to connect each of those pins, using the breadboard to the relevant pins of the Arduino.

Here are the connections for the ultrasonic sensor:

- VCC connects to 5V.
- GND connects to GND (we will use the ground rail on the breadboard, marked with a "-").
- Echo connects to pin 3.
- Trigger connects to pin 4

Now we just mentioned that we will use the ground rail on the breadboard. The rails are the outer two columns of holes that are marked "+" and "-". Power, otherwise known as VCC or V+, is connected to the "+" rail, and ground, otherwise known as GND or V-, is connected to "-". In this project we just use the "-". By connecting the GND from our Arduino to the "-" rail via a jumper cable we create a common ground that any component can safely use.



With the sensor attached, now is the time to connect each of the 9 LEDs to our breadboard. LEDs come with two legs: the longest is the positive leg, commonly known as the Anode; and a shorter leg which is negative/ground and known as a Cathode. When connecting our LEDs to the breadboard, the cathode will be inserted into the same "-" (ground) rail that we used for the sensor. The longer anode leg needs to be inserted into the main breadboard area, so do this for all of the LEDs.

Our LEDs require a resistor in line from the Arduino to the LEDs. We need this to protect the LED from too much current, which can damage or shorten the life of our LEDs. For each of the LEDs use a 220Ω resistor that bridges the central channel and is in line with the LED anode leg. With the resistors inserted now grab some male-to-male jumper cables and wire to the Arduino as follows:

- Red1 = pin 13.
- Red2 = pin 12.
- Red3 = pin 11.
- Yellow1 = pin 10.
- Yellow2 = pin 9.
- Yellow3 = pin 8.
- Green1 = pin 7.
- Green2 = pin 6.
- Green3 = pin 5.

Before applying power double-check all of your connections; the worst thing that can happen is that an LED will pop, but checking your circuit is a good habit to get into. When ready, connect your Arduino to your computer via the USB lead and upload the code to your board via the upload button in the Arduino application. After about 10 seconds your project will come to life and you can move your hand in front of the sensor to trigger the different coloured LEDs. If the code does not auto start, press the reset button on your Arduino.

That's it! You've built your very own distance sensor using less than £10 of parts and around 80 lines of code. You're officially an electronics engineer!

You can find the complete code for this project at our GitHub **https://github.com/lesp/LinuxVoice_Issue11_Arduino_Project** or as a Zip file at **https://github.com/lesp/LinuxVoice_Issue11_Arduino_Project/archive/master.zip**

Les Pounder is a maker and hacker specialising in the Raspberry Pi and Arduino. Les travels the UK training teachers in the new computing curriculum and Raspberry Pi.

LINUX VOICE TUTORIAL

PREY: RECOVER STOLEN DEVICES

MAYANK SHARMA

Lost your laptop? Don a deerstalker and follow our advice to pull a fast one on the perp. The game is afoot!

WHY DO THIS?

- Recovering a device is better than replacing it.
- The software is easy to set up and administer.
- It's also very economical and even has a very usable no-cost plan.

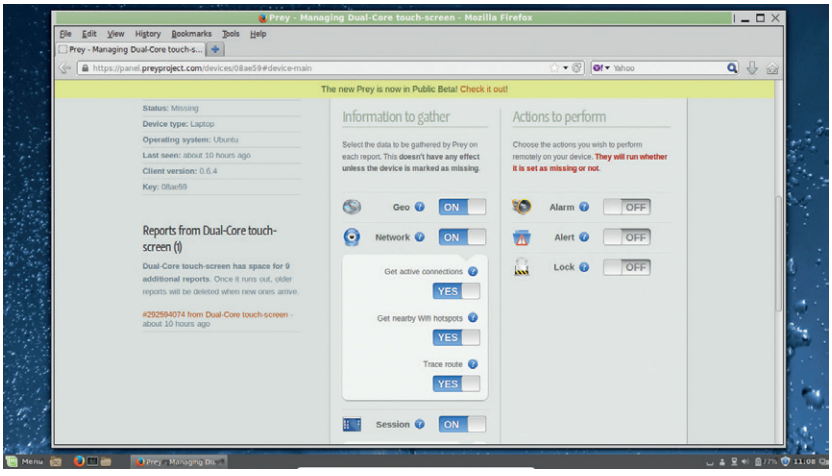
Linux does a wonderful job of insulating your computer from the electronic nasties floating about. But it all comes to naught if you leave it on a bus or lose it in a robbery. The open source *Prey* software helps recover your stolen devices by allowing you to track and control them remotely and make them unusable to anyone who's got them.

Remember, however, that to take advantage of *Prey*'s abilities you'll need to install it before losing control of the laptop. *Prey* installs an agent on your device that runs in the background and periodically sends an HTTP request to check in with its online headquarters on whether it should perform any action or stay asleep. When you lose a device, you mark it as such on *Prey*'s dashboard and the device then starts collecting information to help you track it down.

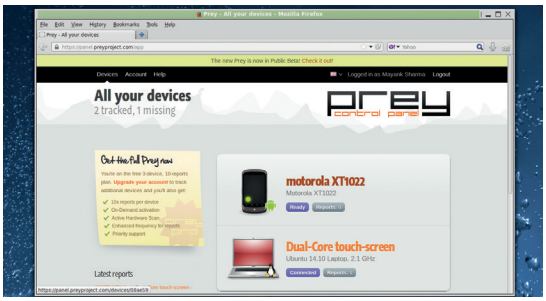
Besides Linux, *Prey* works on several operating systems including Windows, Mac OS X, and even Android and iOS, so you can use it to track laptops and mobile devices as well. You can use it for free to track up to three devices or upgrade to a paid Pro plan starting from $5/month (about £3).

Set up a device

The *Prey* project has pre-compiled binaries for Deb-based distros such as Debian and Ubuntu. To set up *Prey* on these distro, head to preyproject.com and click on the 'Download now' button to grab the binary. You can double-click on the downloaded **.deb** file or use the **sudo dpkg --install prey-*** command to install the *Prey* agent.



The best thing about the web-based control panel is that it allows you to configure the behaviour of Prey on the stolen machine even after it's been pilfered.



To prevent the thief from formatting your laptop, disable booting from removable devices and also lock the BIOS.

After it installs, *Prey* will fire up its graphical configuration tool and asks you to set up a reporting method. The reporting method controls how *Prey* communicates with the devices and reports back to you. The recommended method is to use *Prey*'s web-based control panel, which can be accessed from any machine. You can also optionally run *Prey* in standalone mode, which would require you to set up your own SMTP mail server whose settings you need to specify in prey's config file under **/usr/share/prey**.

Next you need to register with the service. You can do so from within the app or by visiting **preyproject.com**. The *Prey* setup asks you for your name, email address, and a password. After the account has been set up, *Prey* will ask you to add the laptop to the list of tracked devices. It'll automatically pick up the name of the device and its type, which you can edit later from *Prey*'s control panel. For subsequent installations on other devices, select the option to link the device with your existing account.

That's it. You are now ready to set up *Prey*'s behaviour. If it isn't already running, launch the Prey Configurator (which should be under the Administration applications menu), and switch to the Main settings tab. Here you can enable a password-less guest user account to lure whoever is using your stolen device. You should also opt to activate the Wi-Fi autoconnect option, which discreetly connects to the nearest open Wi-Fi hotspot and starts sending you reports.

Configure behaviour

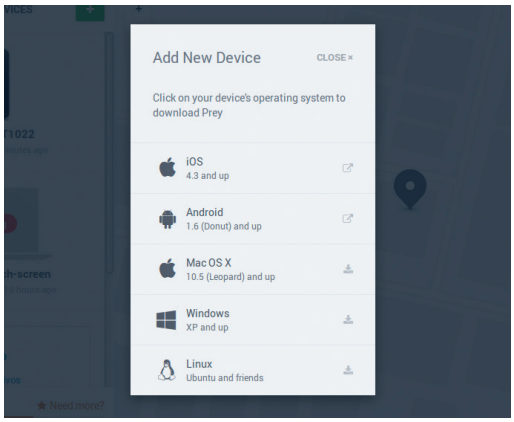
After you've set up your device, you can configure its behaviour via *Prey*'s web-based control panel. The control panel is broken into various sections that control different aspects of the device.

Prey on an Android device

Besides netbooks, laptops, and desktops, *Prey* can also protect mobile devices. To install *Prey* on an Android device, download it from the Google Play Store. Besides Android, the app also runs on iOS and is available on Apple's App Store as well. Once it's installed, hook it to your account if you already have one, or create a new one, just like you do on the laptop version of *Prey*.

After associating it with your account, *Prey* prompts you to activate the *Prey* administrator by locking down the software with a password for extra security. Once activated, whoever's got your device will first have to revoke the privileges provided by the administrator before they can uninstall the software from your phone.

The *Prey* for Android app has the Disable Power Menu option, which when enabled prevents your device from being turned off by disabling its power menu. Also the *Prey* dashboard for the Android device has an additional option. You can toggle the Hide switch, which will then hide the icon for the *Prey* app from the Home screen.



Both the Android and iOS versions are written in their respective platform's native languages.

The main dashboard lists all added devices. Click on a device's name to set it up. Switch to the Configuration section, from where you can alter the name of the device as well as its type in case it wasn't correctly detected. Then move to the Hardware section, which gives you detailed information about the hardware on a particular device including the serial number of the device as well as details about its motherboard and other components, which helps you submit a detailed report to the authorities.

The options under the Main section are separated into two groups; 'Actions to perform' lists the actions that *Prey* will take on your device. Although these actions will be performed irrespective of the device being marked as missing or not, it's best to keep them turned off until the device is actually missing.

Some of these options are designed to dissuade the thief soon after you've lost the device. For example, the Alarm option sounds a loud alarm from your missing device to help you locate it, if it's nearby. Then there's the Alert option, which displays an alert message on the screen on the missing device. If these don't work to discourage the thief, you can use the Lock option to prevent the computer from being used until a password is entered. However, you might not want to lock out the perp as you can trace him better when he's using the laptop.

Keep tabs on your prey

When you lose the laptop, log into *Prey*'s web panel, click on the device that's missing, and use the slider at the top to mark it as such. *Prey* can discreetly gather lots of information about the missing device and its current operator. You can mark all the information you wish to gather from the missing devices' page on the web dashboard.

As soon as the device is brought online, *Prey* can use nearby Wi-Fi access points to interpolate the location on your device and mark it on Google maps. Along with this it also gathers other network-related information such as the public and remote IP address of the network the device is connected to. You can also ask *Prey* to run a traceroute (to **google.com**) from the missing machine through the thief's router. For this to work though make sure you install the traceroute package on the missing laptop before losing it.

You can also ask *Prey* to gather information about the desktop session – including a list of running apps along with a screenshot. Sooner or later you will get a screenshot of him logging into his account on a webmail or some other website. While you won't get his password, you'll be able to clearly see his unique username, using which you can contact him.

If your device has an inbuilt webcam (most laptops and netbooks these days do), *Prey* will also secretly take snapshots of whatever's in front of the webcam. It won't take long before you catch the crook in front of your stolen device. You can set the interval after which *Prey* wakes up and collects the information you have asked it to gather. In the free version, this duration can be between 10 minutes to 50 minutes.

The Pro version allows you to take this down to two minutes or, better still, create a persistent connection with the device. One of the benefits of the Pro version is the On Demand Mode, which brings you reports from a missing device in real-time. In this mode, any changes you make to the configuration of the missing device are triggered instantly if the device is online.

You're all set. All you can do now is wait. As soon as the miscreant goes online with your laptop, the *Prey* client will alert the *Prey* web service. Although we hope you never lose your laptop, in case you do, you are now fully prepared to take on the perp who's got it, and either force him to return your device or collect enough information to build a strong case for the authorities to take action on. Happy hunting.

Mayank Sharma has been tinkering with Linux since the 90s and contributes to a variety of techie publications.