Hacktoberfest 2018

El Dorado, AR

Table of Contents

[Sign Up For Hacktoberfest 3](#_Toc527923879)

[GitHub 3](#_Toc527923880)

[Register for Hacktoberfest 3](#_Toc527923881)

[Development Environment 3](#_Toc527923882)

[Working with GIT 4](#_Toc527923883)

[git clone 4](#_Toc527923884)

[git branch 4](#_Toc527923885)

[git commit 4](#_Toc527923886)

[git push 4](#_Toc527923887)

[git checkout git fetch and git pull 4](#_Toc527923888)

[Lesson 1 – Hello World / Blink 5](#_Toc527923889)

[Lesson 2 – Fade 7](#_Toc527923890)

[Lesson 3 – IR Remote On / Off 9](#_Toc527923891)

[Adding the IR library 9](#_Toc527923892)

[How IR communication works 10](#_Toc527923893)

[IR Module Fritzing 11](#_Toc527923894)

[IR Receiver Code 11](#_Toc527923895)

[Lesson 4 – IR Remote Up / Down 14](#_Toc527923896)

[Lesson 5 – IR Transmitter 15](#_Toc527923897)

[Fritzing 15](#_Toc527923898)

[IR Transmitter Code 16](#_Toc527923899)

# Sign Up For Hacktoberfest

We need to sign up for hactoberfest to participate. Before we can register we need to create an account at GitHub.

## GitHub

GitHub is a git repository host, where people can store revisions of their source code. You can create a new project and share your source code with other developers, or you can contribute with other developers on their projects.

Sign up for a new account at;

<https://github.com/join?source=header-home>

## Register for Hacktoberfest

Hacktoberfest is an annual event attempting to get people involved in developing and contributing to open source projects.

Register to participate at;

<https://hacktoberfest.digitalocean.com/sign_up/register>

## Development Environment

To work on our project, we will be using an Arduino, so we will need to download the Arduino integrated development environment (IDE).

<https://www.arduino.cc/en/Main/Software>

In addition to the IDE, we will need to install a tool to allow us to check in our source code and share it with the rest of the world as an open source project through github. You can download the git client software at.

<https://git-scm.com/downloads>

# Working with GIT

Git is a version control system that was developed to track code changes and allow others to collaborate on the same code base. You can create different branches to help work on new features of the code without effecting the master branch while you work on it. When your code is ready to be put in the main / master branch you can submit a pull – request (PR), to allow the code to be reviewed by others before it is merged into this main / master branch.

Common commands you will need to know are;

## git clone

This is how you get a local copy of a code from a remote repository onto your machine as a local repository.

cd /c/projects

git clone https://github.com/YourGitHubAccount/YourProject.git

## git branch

Used to start a new feature branch for you to begin working on new code separate from the main branch. You will later merge it back into your main branch via a pull request in github.com later, once you are done developing the code. This is also how hactoberfest tracks your contributions. You must have 5 pull request completed in order to complete the challenge.

git checkout -b “Lesson1Blink”

## git commit

Commit a revision of your code changes to your local repository. This is how set “bookmarks” of code at a certain point in time within your development. When you commit code you will be able to provide comments of the work you’ve accomplished.

git commit -m “I added code to make an LED blink”

## git push

Pushes all of the changes you’ve committed locally to the remote branch hosted at github.com.

git push

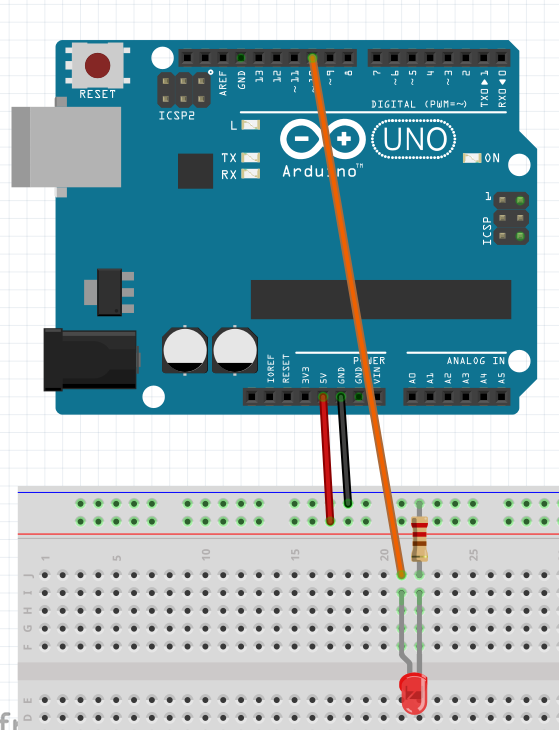
## git checkout git fetch and git pull

Git checkout will switch your local branch to another (such as master). Git fetch, fetches a list of changes that have been made to the remote repository, but does not pull them in yet. Git pull, will actually pull these changes into your local branch and syncs them up.

git checkout master && git fetch -p && git pull

# Lesson 1 – Hello World / Blink

First lesson is to control an LED using a micro controller.



// Our LED is connected to pin 10

const int BLINK\_LED = 10;

/\*\*

\* The setup method gets called only once at the beginning of the application

\*/

void setup() {

// Before we can writ the LED we have to open this PIN for output

pinMode(BLINK\_LED, OUTPUT);

}

/\*\*

\* The loop method gets called over and over again as long as the board has power

\*/

void loop() {

// Turn the LED on

digitalWrite(BLINK\_LED, HIGH);

delay(1000); // Pause for 1000 MS or (1 second)

// Turn the LED off

digitalWrite(BLINK\_LED, LOW);

delay(1000); // Pause for 1000 MS or (1 second)

}

# Lesson 2 – Fade

Add the functionality to fade an LED. We will use this later with the IR remote to turn the brightness up and down.

Add the following constants and variable declarations at the top of the sketch.

const int FADE\_AMOUNT = 5; // How quickly to fade the LED by

int brightness = 0; // Variable to hold the current brightness of the LED start it at 0 or completely off

int direction = 1; // Variable to use for increasing the brightness or decreasing it (0 = down, 1 = up)

Add the following two functions.

/\*\*

\* Increases the brightness of the LED

\*/

void increaseBrightness() {

// Add the fade amount to the brightness to increase it

brightness += FADE\_AMOUNT;

// The LED can only go up to 255 (the maximum value for an 8 bit register)

if (brightness + FADE\_AMOUNT > 255) {

brightness = 255;

}

// Pulse the LED at the given brightness

analogWrite(BLINK\_LED, brightness);

}

/\*\*

\* Increases the brightness of the LED

\*/

void decreaseBrightness() {

// Subtract the fade amount from the brightness to decrease it

brightness -= FADE\_AMOUNT;

// The LED can only go down to 0

if (brightness - FADE\_AMOUNT < 0) {

brightness = 0;

}

// Pulse the LED at the given brightness

analogWrite(BLINK\_LED, brightness);

}

The contents of our main loop should look like the following.

if (direction == 1 && brightness >= 255) {

direction = 0;

}

else if (direction == 0 && brightness <= 0) {

direction = 1;

}

if (direction == 1) {

increaseBrightness();

}

else {

decreaseBrightness();

}

// Wait for 30 ms so we can see it slowly fade up or down

delay(30);

# Lesson 3 – IR Remote On / Off

In this lesson we will add IR functionality to our project so we can control the LED using a remote control.

For this we will take advantage of an open source project another developer created and posted on github.

<https://github.com/z3t0/Arduino-IRremote>

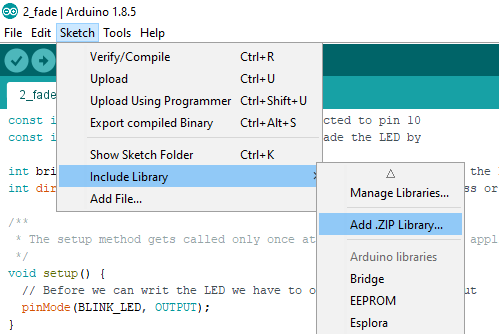
This is a library that was built to take care of some of the more complicated aspects of receiving an IR signal, so we can focus on the application logic without having to worry about the hard details of working with the IR serial protocols.

## Adding the IR library

We will add the library from the zip file in our resources.

Click Sketch->Include Library->Add .ZIP Library

And select the IRremote.zip file.



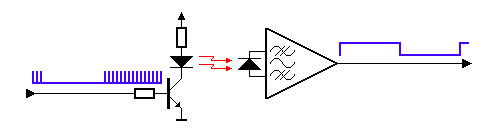
## How IR communication works

Imagine a movie you’ve seen where somebody sends a message using Morris code by “tapping out” a signal. Or sending smoke signals. That’s serial communication. IR is no different, it’s just one end of the signal is flashing a light that is invisible to the human eyes (infrared), and another sensor which is sensing this light as a series of on / off signals representing a unique binary number.

Numbers can be represented in many different ways, we use a base 10 numbering system, but we could represent the number in other bases too.

For example, the number 551,489,775 in our base 10 numbering system would be represented as 20DF10EF in a base 16 numbering system known as hexadecimal. In binary (a base 2 numbering system), it looks like 0010 0000 1101 1111 0001 0000 1110 1111.

This on / off sequence is sent at a specific frequency where it will pulse the LED on and off at specific frequency, where it can then be read back in by another sensor at the other end and interpreted to mean something special (such as turning a TV on or off).



This image was taken from <http://www.sbprojects.com/knowledge/ir/index.php>

In our first example, we will write some code to interpret commands sent from a remote to control an LED on our board. These numbers will be represented in our code as hexadecimal (the Base 16 numbering system where we count from 0 through F before incrementing).

In code we will put 0x in front of our number to designate it as a hexadecimal number. Hex is useful in quickly representing a byte of data. Hex is an shorter way to represent a half a byte (8 bits) of data.

Example..

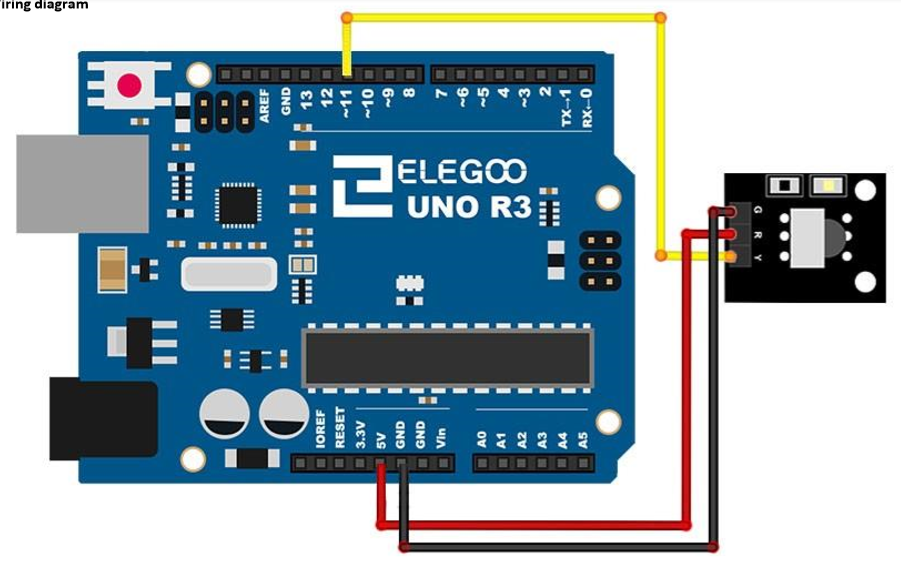
Decimal: 9 = Hex: 9 = Binary: 0101

Decimal: 15 = Hex: F = Binary: 1111

Decimal: 16 = Hex: 10 = Binary: 0001 0000

Decimal: 255 = Hex: FF = Binary: 1111 1111

## IR Module Fritzing



See the PDF in the resources for more conversation on the IR module.

## IR Receiver Code

Include the IRremote library into your project.

#include "IRremote.h"

This will include Ken’s IR library, with several classes and function already pre-built wrapping all the hard stuff up for us so we don’t have to write all of the really hard serial communication code to flash the IR LED.

Add a new constant to represent our GPIO pin that we will use to read the IR data from.

const int IR\_RECEIVER = 11; // Our IR receiver will be connected to GPIO Pin 11

We will now create a couple of new objects from our IR library to help us with the code.

Two of the classes that Ken’s library gives us is 1 for receiving IR signals (IRrecv), and one to decode the data received into numbers that we can match to certain buttons (decode\_results).

IRrecv irReciever(IR\_RECEIVER); // create instance of 'irrecv'

decode\_results irResults; // create instance of 'decode\_results'

We will now need to create a function to toggle our LED from off to on.

/\*\*

\* Turns the LED On or off

\* If it's on already it will turn it off..

\* If it's completely off it will turn it on at full brightness

\*/

void toggleLED() {

if (brightness > 0) {

brightness = 0;

}

else {

brightness = 255;

}

analogWrite(BLINK\_LED, brightness);

}

We will also create a function to handle the IR codes we receive from the remote. We will map all the buttons, we will only wire up the power button right now.

/\*\*

\* This function translates a code received from the remote to actual

\* application logic that we want to perform within our application.

\*/

void translateIR() {

switch(irResults.value) {

case 0xFFA25D: // POWER

toggleLED();

break;

case 0xFFE21D: // FUNC/STOP

case 0xFF629D: // VOL+

case 0xFF22DD: // FAST BACK

case 0xFF02FD: // PAUSE

case 0xFFC23D: // FAST FORWARD

case 0xFFE01F: // DOWN

case 0xFFA857: // VOL-

case 0xFF906F: // UP

case 0xFF9867: // EQ

case 0xFFB04F: // ST/REPT

case 0xFF6897: // 0

case 0xFF30CF: // 1

case 0xFF18E7: // 2

case 0xFF7A85: // 3

case 0xFF10EF: // 4

case 0xFF38C7: // 5

case 0xFF5AA5: // 6

case 0xFF42BD: // 7

case 0xFF4AB5: // 8

case 0xFF52AD: // 9

case 0xFFFFFFFF: // REPEAT

default:

break; // Do nothing

}

delay(100); // Pause for a little bit to "debounce" the remote button press

}

Now in our main loop we will check to see if we have received any IR codes and then act on it accordingly.

/\*\*

\* The loop method gets called over and over again as long as the board has power

\*/

void loop() {

if (irReciever.decode(&irResults)) {

translateIR();

irReciever.resume();

}

}

# Lesson 4 – IR Remote Up / Down

Now that we can interact with the remote, can you figure out how to use the volume up and volume down buttons to dim the LED?

Hint.. Modify the switch statement in the translateIR function to call your increase / decrease brightness functions.

Solution;

case 0xFF629D: // VOL+

increaseBrightness();

break;

case 0xFFA857: // VOL-

decreaseBrightness();

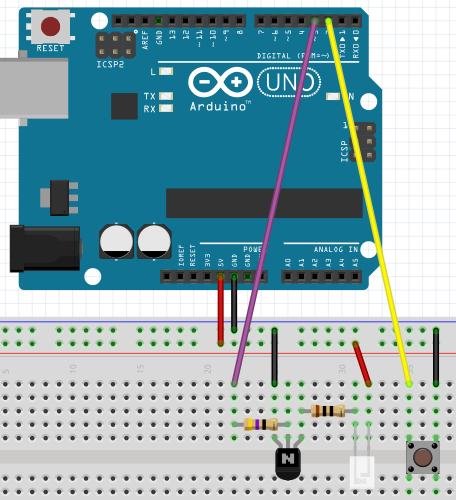
break;

# Lesson 5 – IR Transmitter

Now that we’ve played with receiving IR signals, we will play with transmitting the signals. For this we will use the same open source library developed by Ken Shirriff at <https://github.com/z3t0/Arduino-IRremote>

## Fritzing

For this we will need a button, an IR LED, and with a little bit of circuitry (a 2N4401 transistor and some resistors (10 ohm and 47 ohm), to amplify the LED.



## IR Transmitter Code

When our button is pressed we want to transmit the power signal to the TV. I decoded Jason’s TV remote to send code 0x20DF10EF in NEC format.

This code 0x20DF10EF is a hexadecimal representation of a 32 bit number. In binary it looks like;

0010 0000 1101 1111 0001 0000 1110 1111

Ken’s library takes care of all of this hard work for us, and we can simply send it as a 32 bit NEC code. He wrapped all of his code up in a “class” called IRsend which gives us handy functions to call already pre-built.

At the top of our code we define a constant to represent the pin our button is attached too, and create an instance of IRsend from Ken’s library used to transmit IR signals.

const int BUTTON\_PIN = 2; // Our button is on PIN 2

IRsend irSend; // Create an instance of 'IRsend'

In the setup function we have initialize our “button” to be an input with a normal state of high / with electricity flowing through it. When the button is pushed it will be grounded (or shorted out), bringing the line low. So we set it up as INPUT\_PULLUP.

pinMode(BUTTON\_PIN, INPUT\_PULLUP);

In our main loop we check for the button to be pushed (showing it’s been grounded as LOW). And if it has we will send the signal. We also have to have to pause for a couple of seconds to debounce the button. Because our loop runs so fast that when we push the button, we could send the “power” code multiple times, so we pause for just a couple of seconds.

if (digitalRead(BUTTON\_PIN) == LOW) {

// Send the power button code for the TV

irSend.sendNEC(0x20DF10EF, 32);

delay(2000);

}