Project: Add modern automation to ‘80s GE Low Voltage Lighting Relay system.

Materials:

Concentrated panel where relay circuits are available.

Relay map for your home if available.

Raspberry Pi

> I used a Pi 3+ for my project, but this is lightweight enough that anything that can run Django should do the trick.

USB Relay Cards

> You’ll need enough relays available to cover twice the number of relays/light switches you have; this is because On and Off are activated through different relays.
>
> I installed four SainSmart 16 relay USB cards.
>
> https://www.amazon.com/SainSmart-16-Channel-9-36V-Relay-Module/dp/B0793MZH2B

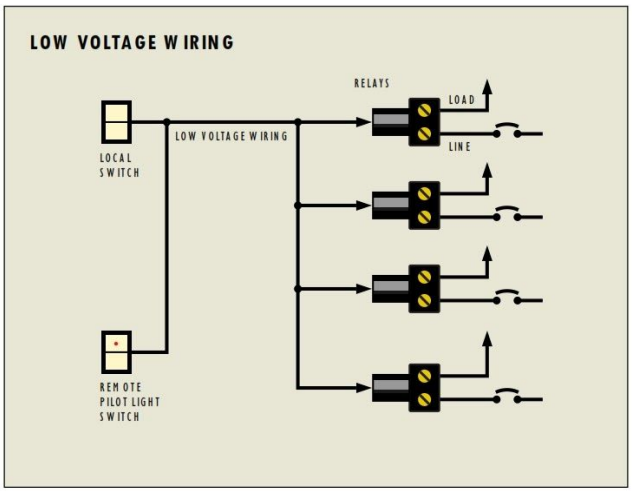
Ethernet cable spool or scrap wire.

Larger gauge wire for power pulls.

The code I used for controlling the cards and acting as a frontend is uploaded to my github here: https://github.com/kjatar/RelayControllers

You'll need to customize it for your personal uses!

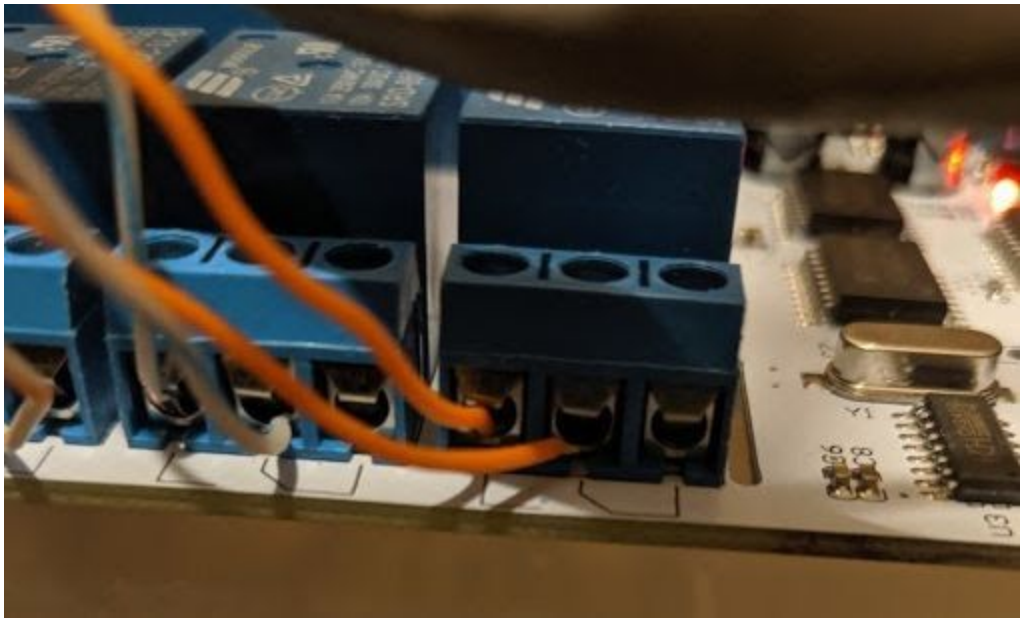
Process:
One of the interesting features of the GE LV system is that it allows multiple switches to be mapped to a single relay. This allowed for central controllers and upper floor/lower floor controls in the original system, but it ALSO means that we can integrate this with our modern system by setting up the USB relay boards as "phantom switches".



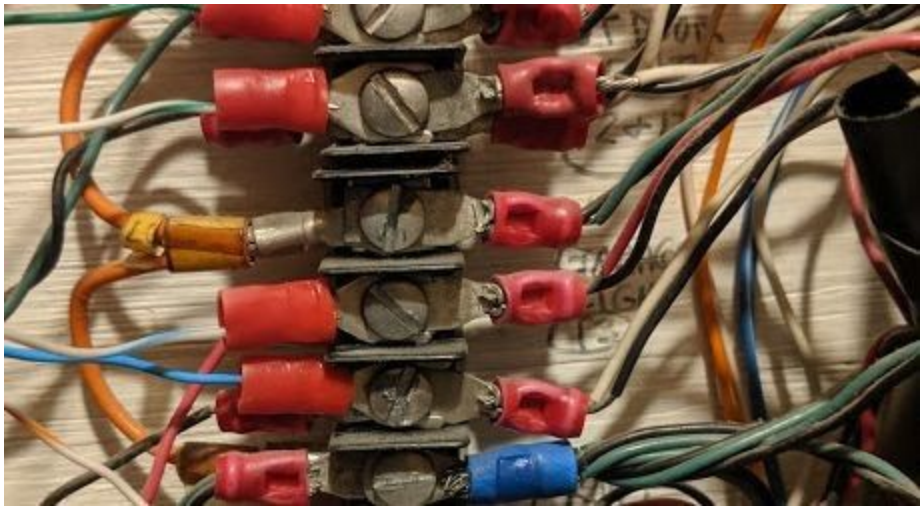
So, the first thing to do is to identify the incoming power source. In mine, I was able to trace this to a power source directly plugged into the wall, but you should be able to find it by finding the segment that gets forked off across many different end circuits; shorting that connector with one of the relay-end connectors will flip the relevant relay in your house. Identifying one near your workspace for ease of testing will be useful!

Once you've identified the power source, tap it off with a thicker power cable and bring it down towards where you're installing your Pi and cards. At that point, splice it with a number of individual ethernet strands. Each of those strands is going to go into one half of a relay on your relay card. The power being supplied to the GE LV relays is supposed to be 24v AC power, though in some places 12v DC was used.

Each relay will have three slots. One is always connected, one is connected when the relay is off, one is connected when the relay is on. I would recommend putting the incoming power into the segment that is only connected when the relay is on, but it can also be wired into the always connected segment. Whatever you choose, be consistent! This will make it easier for troubleshooting.



For each of the GE LE relays, there will be one board relay for the on position and one board relay for the off position. I was lucky enough to have a map of the relay wiring installed in my house to go off of, so I had to do less identification during the process, but with a helper, you can short the power directly to each set of wires and see what relay in your house triggers. Document this in a spreadsheet, you'll need this later for the frontend.



Once you know which wires go to which relay and position, it's time to use the interior ethernet strands again. Splice or screw one end of each strand onto the "from switch" end of the wall wiring, then to one of the empty slots on one of the relay boards, such that when the relay activates power would flow through the strand.

Repeat for each and every relay, both on and off.

The hardware side of this is done outside of troubleshooting.

On to software!

The first key is to ensure that each of the relays and relay boards works. If you're using the same SAINsmart 16 relay boards that I used, you can test this easily enough using the "test" method in MyRelays.py; this will go through and trigger each relay for a quarter of a second. For production purposes, the amount of time each relay is triggered for can be lowered, but this will give a good initial evaluation.

I didn't bother to fully decode the command structure, but I have included it; short version, to activate one of these relays in your own code, I used the serial library in python and initialized like so:

```
Import serial
ser = serial.Serial()
ser.baudrate = 9600
ser.bytesize = serial.EIGHTBITS #number of bits per bytes
ser.parity = serial.PARITY_NONE #set parity check: no parity
ser.stopbits = serial.STOPBITS_ONE #number of stop bits
ser.timeout = 2            #timeout block read
ser.xonxoff = False     #disable software flow control
ser.rtscts = False     #disable hardware (RTS/CTS) flow control
ser.dsrdtr = False       #disable hardware (DSR/DTR) flow control
ser.writeTimeout = 2     #timeout for write
```

An example of the code to trigger one of the relays looks like this:

```
if sys.argv[1] == "1":
   ser.port = "/dev/ttyUSB0"
   ser.open()
   ser.write(bytearray.fromhex("3A 46 45 30 35 30 30 30 30 46 46 30 30 46 45 0D 0A"))
   sleep(0.25)
   ser.write(bytearray.fromhex("3A 46 45 30 35 30 30 30 30 30 30 30 30 46 44 0D 0A"))
```

Effectively, if the script is called with an argument of "1", i.e. trigger the first relay, the code will first set the port to /dev/tty/USB0, which is relay card 1 of 4; second, it opens a channel to write; it writes the command string telling that card to close relay 1 on the card, waits a quarter second, then writes a second string telling it to open relay 1.

If the relay number is over 16, the script will move on to /dev/tty/USB1, USB2, or USB3 respectively.
Yes, I could have simplified this script using modulo math, I haven't gotten to it yet, go away.

I've heard some concerns about whether the cards will initialize in the same order each time or if that presents a race condition, but after extensive testing they have consistently come up in the order expected and gotten labeled properly under /dev/tty.

The frontend site is very simple; it's just a static site running a simple javascript to make a url request of /relays/<relaynumber>/ to localhost with no reload on button press; the url request is parsed by the Django server to pull out the <relaynumber> parameter and then run the MyRelays script with that input.

This also means you can set up either custom commands that do, for example, groups of lights. You can also set up scheduled tasks or cron jobs to trigger lights on or off at specific times.

I recommend adding input validation to either the Django server (better) or the MyRelays script (okay but not as good).

Don't leave this code public facing. Restrict for internal network use at your house.

Future Ideas:
One thing I wasn't able to implement in v1 here is a statefulness system. Since the main switches still work, there's no way to tell what state a relay is in without actually looking at the light. I'd like to integrate some sort of light sensors to be able to provide that feedback to the system, but that's a second level of complexity I just haven't gotten to yet.