Linux

- 5.1 Surround Sound on 3 3.5mm Ports Motherboard
- Default file and directory permissions in Linux
- Overclocking multiple headless Nvidia GPU for CUDA Cryptocurrency mining on Linux
- Wireguard server and peer configuration (Archlinux / Debian / Ubuntu)
- Setting up linuxserver/netbootxyz docker image and dnsmasq DHCP in proxy mode when your main router has locked DHCP settings
- Port forwarding through a VPS with the help of Wireguard and iptables

5.1 Surround Sound on 3 3.5mm Ports Motherboard

Context

I've recently purchased an MSI B450M BAZOOKA PLUS motherboard that supports 7.1 surround sound on their measely 3 3.5mm ports.

I have a 5.1 surround sound setup with my Logitech Z506 set that was previously working just plugging in to my old motherboard Asus P8P67 PRO which had 6 3.5mm ports.

Instructions

Graphical

1. Install the `hdajackrestask` tool on your system (Archlinux: `pacman -S alsa-tools`)
2. Run in privileged mode using sudo or polkit (GKSU: `gksu hdajackretask`)
3. Plug your 3 3.5mm audio jacks according to this image, colours should be roughly the same, alternatively take reference from the label (mic, L-Out and L-In) Image not found or type unknow
   3.5mm audio jacks layou
4. Use these settings for 5.1 surround sound setup Image not found or type unknown
   hdajackretask settings for 5.1 surround sound
5. Click on `Install boot override` and reboot

CLI

For 5.1 surround sound, as root run:

```
# echo "options snd-hda-intel patch=hda-jack-retask.fw,hda-jack-retask.fw,hda-jack-
```

```
retask.fw,hda-jack-retask.fw" > /etc/modprobe.d/hda-jack-retask.conf
```

Default file and directory permissions in Linux

Context

Just yesterday night while I was configuring Grav, I realised some of the permissions were not setting right and I was unable to modify files created by Grav via a privileged user with SFTP.

Looking into it, I checked the properties via `ls -l` and sure enough:

```
$ ls -l /-snip-grav-parent-directory-/
total -snip-
drwxr--r-x 13 -snip-grav-user- -snip-grav-group- 4096 Aug  6 08:00 -snip-grav-folder-
```

Since the files were owned by the Grav user, while my user was in the Grav group. I was unable to edit the files due to the absense of write permissions for the group clause.

Simple chmod

At first, running a simple recursive chmod everytime I met the issue was fine as normally my web files were not created by the Grav user as I mainly host static pages before:

```
$ chmod -R g+w /-snip-grav-directory-/
```

Default groups and permissions

But there was two problems, the first problem is that new files created by my privileged user had the group not set to the Grav user and thus not allowing Grav to modify it too. I didn't wish to add the grav user to any groups so I had to resort to recursively adding the *setgid* bit on the directory:

```
$ chmod -R g+s /-snip-grav-directory-/
```

That solves the first problem by having new files automatically set its' group to the parent directory whenever they are created.

The other problem was there was no default permissions and new files were created were not allowed to be modified by my privileged user. That's where this nifty tool comes in, acl (Access Control Lists) which extends the basic chmod, chown and chgrp commands.

This tool requires you to add `acl` your filesystem mount option to always apply if it's not already set, you can check it by running this and seeing if `acl` is returned:

```
$ tune2fs -l /dev/sdXY |  grep "Default mount options:"
```

Most filesystems turn it on by default, but if it does not, proceed to add it into `/etc/fstab`.

After that it's just a simple command as this to set default permissions for the directory and the files

```
$ setfacl -R -d -m g::rwX -snip-grav-directory-
```

- `setfacl` - set acl command
- `-R` - recursive
- `-d` - target **d**efault permissions
- `-m` - **m**odify
- `g::rwX` - target **g**roup with **r**ead **w**rite and only directories should be set with e**X**ecutable

A confirmation with `getfacl`:

```
$ getfacl -snip-grav-directory-
getfacl -snip-grav-directory-
# file: -snip-grav-directory-
# owner: -snip-grav-user-
# group: -snip-grav-group-
# flags: -s-
user::rwx
group::rwx
other::r-x
default:user::rwx
default:group::rwx
default:other::r-x
```

Once you see `# flags: -s-` and `default:group::rwx`, you are all set and future new directories and files inside will be set with the default permissions automatically.

You may want to run `chmod -R g+X` the directory after the above step to apply the permissions to the existing files.

Pointers from mistakes I made

- Do not forget to set the executable permission for directories, if it is missing the user will not be able to access it, note the pointer below when needing to fix this.
- Do not ever run `chmod -R g+x` as it would set all files executables too which is a security risk, instead use `chmod -R g+X` with capital `X` which only sets directories with executable, same thing for `acl`.

Further reading and references

- Archlinux Wikipedia article on ACL
- Archlinux Wikipedia article on file permissions and attributes

Overclocking multiple headless Nvidia GPU for CUDA Cryptocurrency mining on Linux

Setup

```
$ lspci | grep VGA
01:00.0 VGA compatible controller: NVIDIA Corporation GP104 [GeForce GTX 1070] (rev a1)
02:00.0 VGA compatible controller: NVIDIA Corporation GP104 [GeForce GTX 1070] (rev a1)
03:00.0 VGA compatible controller: NVIDIA Corporation GP104 [GeForce GTX 1070] (rev a1)
04:00.0 VGA compatible controller: NVIDIA Corporation GP104 [GeForce GTX 1070] (rev a1)
06:00.0 VGA compatible controller: NVIDIA Corporation GP104 [GeForce GTX 1070] (rev a1)
07:00.0 VGA compatible controller: NVIDIA Corporation GP104 [GeForce GTX 1070] (rev a1)
```

6x Gigabyte GTX 1070 G1 Gaming (7.93 GB VRAM) pool mining Ethereum.

Steps

Firstly we will have to one-time initialize /etc/X11/xorg.conf by using Nvidia's xconfig tool including a cool-bits value of 28.

```
# nvidia-xconfig --allow-empty-initial-configuration --enable-all-gpus --cool-bits=28
```

Afterwards, we can start a Xorg server headlessly as Nvidia's setting tool requires it to be up.

```
# X :0 & # display :0
```

Since I am running multiple cards, I would require a loop to go through all of it. Currently these values works best for the cards I used.

- -200 core clock
- +1200 memory clock

```
# export DISPLAY=:0
# IFS=$'\n'
# for gpu in $(nvidia-smi -L); do
    id=$(echo ${gpu} | sed -e 's/GPU //g' -e 's/:.*//g')
    nvidia-settings -a [gpu:${id}]/GPUPowerMizerMode=1 -a [gpu:${id}]/GPUGraphicsClockOffset[3]=-200 -a [gpu:${id}]/GPUMemoryTransferRateOffset[3]=1200
done
```

As a final step, I would apply a -40 max watts undervolt as root.

```
# nvidia-smi -pl 140
```

Optionally, since we do not need the Xorg server to be running anymore, we can kill it

```
# killall Xorg
```

These configuration alone increased ~5MH/s per card, resulting in a 25-30MH/s overall increase!

Automated bash script

```
#!/usr/bin/env bash
sudo nvidia-smi -pl 140
sudo X :0 &
sleep 5
export DISPLAY=:0
IFS=$'\n'
for gpu in $(nvidia-smi -L); do
    id=$(echo ${gpu} | sed -e 's/GPU //g' -e 's/:.*//g')
    sudo nvidia-settings -a [gpu:${id}]/GPUPowerMizerMode=1 -a [gpu:${id}]/GPUGraphicsClockOffset[3]=-200 -a [gpu:${id}]/GPUMemoryTransferRateOffset[3]=1200
done
sudo killall Xorg
sudo systemctl start ethminer # or your own command to start your own miner
```

Wireguard server and peer configuration (Archlinux / Debian / Ubuntu)

Introduction

Wireguard is a new VPN tool that is vastly easier to setup than the popular alternative OpenVPN. Also reports state that it is also superior in speed and reliability.

Setup

General

Installation

Most package managers should have the required packages named

`wireguard-tools wireguard-dkms`

Install them on both your server and client(s).

Note: very soon, Wireguard will become baked into the Linux kernel by default and `wireguard-dkms` will not be needed anymore.

Generation of private and public key pair

```
$ (umask 077 && printf "[Interface]\nPrivateKey = " | sudo tee /etc/wireguard/wg0.conf > /dev/null)
$ wg genkey | sudo tee -a /etc/wireguard/wg0.conf | wg pubkey | sudo tee /etc/wireguard/publickey
```

Replace `wg0` with your desired network device id throughout the article if needed.

This generates a private key and automatically inserts as a configuration line to `/etc/wireguard/wg0.conf` and a public key saved to `/etc/wireguard/publickey` automatically. Run it on both your server and client(s) respectively.

Server

Edit `/etc/wireguard/wg0.conf`

```
[Interface]
# Private key, automatically generated by above command on the server (should be only 44 characters as of writing)
PrivateKey = -auto generated-

# Private IPv4 and IPv6 address of Server for peers to communicate with when connected, you can replace `123.210` and `123:210` with anything you like throughout the article
Address = 10.123.210.1/24,fd00:123:210::1/112

# Listen port, can be any port you like including 53 if you don't use it for DNS. Must be the same throughout the article.
ListenPort = 51820

# Setup IPv4 and IPv6 iptables to forward the network of peers through the server, not required if only a LAN connection is required (optional)
PostUp = iptables -A FORWARD -i %i -j ACCEPT; iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE; ip6tables -A FORWARD -i %i -j ACCEPT; ip6tables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
PostDown = iptables -D FORWARD -i %i -j ACCEPT; iptables -t nat -D POSTROUTING -o eth0 -j MASQUERADE; ip6tables -D FORWARD -i %i -j ACCEPT; ip6tables -t nat -D POSTROUTING -o eth0 -j MASQUERADE

# Save the configuration to the file on every shutdown, personally I prefer it off because I find it easier to edit the configuration directly rather than to rely on tools
SaveConfig = false



# CLIENT 1
[Peer]
```

```
# Public key of the peer, generated by the above command on the peer (also should be only 44 characters as of writing)
PublicKey = -auto generated and copied here-

# Allow IPv4 and IPv6 range from 10.123.210.1-10.123.210.254 and fd00:123:210::1-fd00:123:210::ffff respectively
AllowedIPs = 10.123.210.0/24,fd00:123:210::0/112



# CLIENT 2
[Peer]
# Public key of the peer, generated by the above command on the peer (also should be only 44 characters as of writing)
PublicKey = -auto generated and copied here-

# Allow IPv4 and IPv6 range from 10.123.210.1-10.123.210.254 and fd00:123:210::1-fd00:123:210::ffff respectively
AllowedIPs = 10.123.210.0/24,fd00:123:210::0/112

# ... More peers if required ...
```

Additional step to allow forwarding (optional)

```
$ echo -e "net.ipv4.ip_forward=1\nnet.ipv6.conf.all.forwarding=1" | sudo tee -a /etc/sysctl.d/99-sysctl.conf
$ sudo sysctl -p
```

Start the server

```
$ sudo systemctl enable --now wg-quick@wg0
```

Client(s)

Edit `/etc/wireguard/wg0.conf`

```
[Interface]
# Private key, automatically generated by above command on the client (should be only 44 characters as of writing)
PrivateKey = -auto generated-
```

```
# Private IPv4 and IPv6 address of client, must be static IP (no clashes) because there is no DHCP provided by Wireguard as of writing. Change the `2` to an incremental number for every client
Address = 10.123.210.2/32,fd00:123:210::2/128

# DNS server to use, currently set to Cloudflare
DNS = 1.1.1.1



# SERVER
[Peer]
# Public key of server, generated by the above command on the server (only 44 characters as of writing)
PublicKey = -auto generated and copied here-

# Public IP of server and port configured in the server
Endpoint = -public key of server-:51820

# IP ranges Wireguard will listen on and forward
# AllowedIPs = 10.123.210.0/24,fd00:123:210::0/112 # ROUTE ONLY VIRTUAL PRIVATE NETWORK TRAFFIC
AllowedIPs = 0.0.0.0/5, 8.0.0.0/7, 11.0.0.0/8, 12.0.0.0/6, 16.0.0.0/4, 32.0.0.0/3, 64.0.0.0/2, 128.0.0.0/3, 160.0.0.0/5, 168.0.0.0/6, 172.0.0.0/12, 172.32.0.0/11, 172.64.0.0/10, 172.128.0.0/9, 173.0.0.0/8, 174.0.0.0/7, 176.0.0.0/4, 192.0.0.0/9, 192.128.0.0/11, 192.160.0.0/13, 192.169.0.0/16, 192.170.0.0/15, 192.172.0.0/14, 192.176.0.0/12, 192.192.0.0/10, 193.0.0.0/8, 194.0.0.0/7, 196.0.0.0/6, 200.0.0.0/5, 208.0.0.0/4, ::/0, 10.123.210.0/32 # ROUTE ALL INTERNET TRAFFIC LESS LAN THROUGH

# Constant pings to keep the connection alive and not time out on inactivity
PersistentKeepalive = 25
```

Connect to server

```
$ sudo wg-quick up wg0
```

Connection information

You can run these commands to check the connection

```
```

```
$ sudo wg
$ ping 10.123.210.1
```

Disconnect from server

```
$ sudo wg-quick down wg0
```

Extra information

networkmanager-wireguard

If you use NetworkManager (especially `nm-applet`) you can install `networkmanager-wireguard` or `networkmanager-wireguard-git` (AUR) for Wireguard capabilities and configuration.

Forward other UDP ports to Wireguard port with iptables

On the server:

```
$ sudo iptables -t nat -I PREROUTING -i eth0 -p udp -m multiport --dports 53,80,123,161,443 -j REDIRECT --to-ports 51820
```

To disable:

```
$ sudo iptables -t nat -D PREROUTING -i eth0 -p udp -m multiport --dports 53,80,123,161,443 -j REDIRECT --to-ports 51820
```

You can add it to `PostUp` and `PostDown`. Don't forget `ip6tables` if needed.

More reading

https://github.com/pirate/wireguard-docs#Interface

Setting up linuxserver/netbootxyz docker image and dnsmasq DHCP in proxy mode when your main router has locked DHCP settings

Prerequisites

1. `Docker` is installed
2. `dnsmasq` is installed

Steps

1. Start Docker image from linuxserver.io

Head down to linuxserver/netbootxyz and follow the instructions to get a netboot.xyz container up and running.

2. Configuration

/etc/dnsmasq.conf

Full

```
port=0
interface=eth0
bind-dynamic
log-dhcp
dhcp-authoritative
dhcp-range=192.168.1.0,proxy,255.255.255.0
pxe-service=x86PC, "NETBOOT (BIOS)", "netboot.xyz.kpxe", 192.168.1.253
pxe-service=X86-64_EFI, "NETBOOT (EFI)", "netboot.xyz.efi", 192.168.1.253
```

Explanations

- `port=0` - **(optional)** disable DNS server if not in use
- `interface=eth0` - **(optional)** only listen on interface `eth0`, repeat this line for more interfaces, remove for all interfaces
- `log-dhcp` - **(optional)** log DHCP requests
- `bind-dynamic` - **(optional)** binds to new interfaces added after start
- `dhcp-authoritative` - prioritize this DHCP server in a network
- `dhcp-range` - replace `192.168.1.0` and `255.255.255.0` according to your IP range and subnet, set to `proxy` mode to proxy your original DHCP server
- `pxe-service=x86PC` for BIOS systems, replace `192.168.1.253` with the IP of the machine hosting netbootxyz
- `pxe-service=X86-64_EFI` for UEFI systems, replace `192.168.1.253` with the IP of the machine hosting netbootxyz
- `enable-tftp` is **not included** as it already hosted by the Docker container

3. Start your services

Port forwarding through a VPS with the help of Wireguard and iptables

Context

Recently I wanted to host a Minecraft server on a network that I can not port forward with. Although this can be done with an SSH tunnel too, I believe Wireguard will give a slightly better performance overall.

I will not be going through Wireguard setup in this page, but I did cover it before here: Wireguard server and peer configuration (Archlinux / Debian / Ubuntu). Instead I will just be going through the iptables portion.

Commands

Add

```
# iptables -t nat -A PREROUTING -p tcp -d 123.123.123.123 --dport 25565 -j DNAT --to
10.123.123.123:25565
# iptables -A FORWARD -p tcp -d 10.123.123.123 --dport 25565 -j ACCEPT
# iptables -t nat -o wg0 -A POSTROUTING -j MASQUERADE
```

Delete

```
# iptables -t nat -D PREROUTING -p tcp -d 123.123.123.123 --dport 25565 -j DNAT --to
10.123.123.123:25565
# iptables -D FORWARD -p tcp -d 10.123.123.123 --dport 25565 -j ACCEPT
```

```
# iptables -t nat -o wg0 -D POSTROUTING -j MASQUERADE
```

- Change `123.123.123.123` to your external facing server's public IP address
- Change `10.123.123.123` to the server's Wireguard IP address
- Change all instances of `25565` to a port you wish to forward
- Change `wg0` to your Wireguard interface name