Software & packages

- Image manipulation
- yt-dlp
- Python packages and commands
- spotdl
- ffmpeg
- IPFS
- Wireguard
- Node & NVM
- Syncthing
- Screen
- Tailscale
- smb / samba
- autofs

Image manipulation

Compression

Install `sudo apt-get install imagemagick`

Compress command `mogrify -quality 80% *.jpg`

note: this will overwrite images

yt-dlp

Download video as mp3

```
yt-dlp -x --audio-format mp3 URL_HERE
```

Convert webm to mp4

```
ffmpeg -i input.webm -c:v libx264 -preset slow -crf 22 -c:a aac -b:a 128k output.mp4
```

Download video as mp4

```
yt-dlp -f "bestvideo[ext=mp4]+bestaudio[ext=m4a]/best[ext=mp4]/best" SOURCE_URL
```

Convert video codec to x264

```
ffmpeg -i input.mp4 -c:v libx264 -crf 23 -preset medium -c:a aac -b:a 192k output.mp4
```

Python packages and commands

Managing Virtual Envrionments

Create a new venv `python3 -m venv /path/to/new/virtual/environment`

Activate the new venv Linux / Mac `source myenv/bin/activate`

Activate an envrionment on windows `myenv\Scripts\activate`

Exit the venv `deactivate`

Installing and uninstalling packages

Installing a package `pip install xyz`

Uninstalling a package `pip uninstall xyz`

Useful packages

- aqlalchemy - SQL Alchemy
- snowflake-sqlalchemy - SQL Alchemy for snowflake

spotdl

Install & Setup

> Best to load a venv first

Installing spotdl `pip install spotdl`

Open the config file at `.spotdl/config.json` and update "output" to be `"{album}/{title}.{output-ext}"`

Updating

`pip install --upgrade spotdl`

ffmpeg

Installing

```
sudo apt install ffmpeg
```

Compress a video to h265

```
ffmpeg -i example_h264.mp4 -c:v libx265 -crf 24 -preset fast -c:a aac -b:a 128k example_h265.mp4
```

CRF Info

For H.264 (x264):

The typical CRF range is 18-28.

CRF 18: Near lossless quality (very high quality, larger file size).

CRF 23: Default value, good balance between quality and file size.

CRF 28: Noticeable quality loss, much smaller file size.

Overwright a videos audio with a new audio track

```
ffmpeg -i video_to_overwright.mkv -i audio_input.mp3 -c:v copy -map 0:v:0 -map 1:a:0 output.mp4
```

MP4 to MP3

```
ffmpeg -i filename.mp4 filename.mp3
```

Change video format

```
ffmpeg -i input.m4v -c copy output.mp4
```

IPFS

Installation for Linux

1 - Download the package

```
wget https://dist.ipfs.tech/kubo/v0.31.0/kubo_v0.31.0_linux-amd64.tar.gz
```

2 - Unzip the file

```
tar -xvzf kubo_v0.39.0_linux-amd64.tar.gz
```

3 - Move into the kubo folder

```
cd kubo
```

4 - Run the install script

```
sudo bash install.sh
```

5 - Test the installation worked

```
ipfs --version
```

Editing the config

Show Config:

```
ipfs config show
```

Change connection count

```
ipfs config Swarm.ConnMgr.HighWater 1000 --json
ipfs config Swarm.ConnMgr.LowWater 500 --json
```

Enable filestore (Allow import with no copy)

```
ipfs config --json Experimental.FilestoreEnabled true
sudo systemctl restart ipfs
ipfs config Experimental.FilestoreEnabled
```

IPFS as a service

Open a new file at `/etc/systemd/system/ipfs.service`

```
[Unit]
Description=IPFS daemon
After=network.target

[Service]
ExecStart=/usr/local/bin/ipfs daemon
Restart=on-failure
User=conor
Group=conor
Environment=IPFS_PATH=/home/conor/.ipfs

[Install]
WantedBy=default.target
```

`WantedBy=default.target` means the service will start up at boot

get it to start at boot

```
sudo systemctl enable ipfs
```

and start it

```
sudo systemctl start ipfs
```

Usage

Check filestore is enabled and working

```
ipfs config Experimental.FilestoreEnabled
```

List files in filestore

```
ipfs filestore ls
```

Import files to filestore

```
ipfs add --nocopy --recursive --cid-version=1 /path/to/your/folder
```

Wireguard

Setup Script

```
#!/bin/bash

# Checks to see if script is being run as root
if [ "$EUID" -ne 0 ]; then
  echo "Please run as root"
  exit
fi

sudo apt install wireguard

# Deletes keys in case this script is being run again
rm -f /etc/wireguard/*

# Create necessary directories, including parents
mkdir -p /etc/wireguard/client



# Create and store server private and public key
wg genkey | tee /etc/wireguard/server_priv.key | wg pubkey | tee /etc/wireguard/server_pub.key

# Get the server public and private key as well as the network interface
server_priv_key=$(cat /etc/wireguard/server_priv.key)
server_pub_key=$(cat /etc/wireguard/server_pub.key)
network_interface=$(ip -o -4 route show to default | awk '{print $5}')



# Client
# Create client public and private key and store it in vars for later
wg genkey | tee /etc/wireguard/client/client_priv.key | wg pubkey | tee /etc/wireguard/client/client_pub.key
client_priv_key=$(cat /etc/wireguard/client/client_priv.key)
client_pub_key=$(cat /etc/wireguard/client/client_pub.key)
```

```
# Create initial client config
echo "[Interface]
PrivateKey = $client_priv_key
Address = 10.0.0.2/24

[Peer]
PublicKey = $server_pub_key
Endpoint = servername_or_ip:51820
AllowedIPs = 0.0.0.0/0
" > /etc/wireguard/client/wg0.conf



# Create the config file for wireguard
echo "[Interface]
Address = 10.0.0.1/24
SaveConfig = true
ListenPort = 51820
PrivateKey = $server_priv_key
PostUp = iptables -A FORWARD -i %i -j ACCEPT; iptables -t nat -A POSTROUTING -o $network_interface -j
MASQUERADE
PostDown = iptables -D FORWARD -i %i -j ACCEPT; iptables -t nat -D POSTROUTING -o $network_interface -j
MASQUERADE

[Peer]
PublicKey = $client_pub_key
AllowedIPs = 10.0.0.2/24
" > /etc/wireguard/wg0.conf

# Change permissions so only root can access the files
chmod 600 /etc/wireguard/server_priv.key
chmod 600 /etc/wireguard/wg0.conf

# Allow 51820 through ufw
ufw allow 51820/udp

# Start wireguard and set it to auto start on boot
wg-quick up wg0
systemctl enable wg-quick@wg0
```

Setup - Manual

1 - Update system and install wireguard

```
apt update && apt upgrade && apt install wireguard
```

2 - Make the necessary folders

```
mkdir -p /etc/wireguard/client
```

3 - Create the server's public and private keys

```
wg genkey | tee /etc/wireguard/server_priv.key | wg pubkey | tee /etc/wireguard/server_pub.key
```

4 - Find the servers network interface

```
ip -o -4 route show to default | awk '{print $5}
```

5 - Nano into

```
/etc/wireguard/wg0.conf
```

(Make sure to add network interface and server private key)

```
[Interface] Address = 10.0.0.1/24
SaveConfig = true
ListenPort = 51820
PrivateKey = SERVER_PRIVATE_KEY
PostUp = iptables -A FORWARD -i %i -j ACCEPT; iptables -t nat -A POSTROUTING -o NETWORK_INTERFACE_HERE -j MASQUERADE
PostDown = iptables -D FORWARD -i %i -j ACCEPT; iptables -t nat -D POSTROUTING -o NETWORK_INTERFACE_HERE -j MASQUERADE
```

6 - Update file permissions so only root can read the config file and private key

```
chmod 600 /etc/wireguard/server_priv.key chmod 600 /etc/wireguard/wg0.conf
```

7 - Allow the vpn port through UFW

```
ufw allow 51820/udp
```

8 - Start wireguard and set it to auto start at boot

```
wg-quick up wg0 systemctl enable wg-quick@wg0
```

9 - Create the client keys

```
wg genkey | tee /etc/wireguard/client/priv.key | wg pubkey | tee /etc/wireguard/client/pub.key
```

10 - Create the client config file (copy to client device)

```
[Interface] PrivateKey = CLIENT_PRIVATE_KEY
Address = 10.0.0.2/24 [Peer]
PublicKey = SERVER_PUBLIC_KEY
Endpoint = SERVER_IP_OR_DOMAIN:51820
AllowedIPs = 0.0.0.0/0
```

11 - Add peer info to the server config

```
[Interface] Address = 10.0.0.1/24
SaveConfig = true
ListenPort = 51820
PrivateKey = SERVER_PRIVATE_KEY
PostUp = iptables -A FORWARD -i %i -j ACCEPT; iptables -t nat -A POSTROUTING -o NETWORK_INTERFACE_HERE -j MASQUERADE
PostDown = iptables -D FORWARD -i %i -j ACCEPT; iptables -t nat -D POSTROUTING -o NETWORK_INTERFACE_HERE -j MASQUERADE

# ================
[Peer]
PublicKey = PUBLIC_KEY
AllowedIPs = 10.0.0.2/24
```

Setup with SeaBee's setup script

```
wget https://raw.githubusercontent.com/seabee33/wireguard_helper/refs/heads/main/wg_helper.py && chmod +x wg_helper.py && sudo python3 wg_helper.py
```

Auto start at boot

1 - ensure the client config is at `/etc/wireguard/wg0.conf`
2 - enable it to start at boot with `sudo systemctl enable wg-quick@wg0`

Node & NVM

1 - install node version manager (NVM)

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.3/install.sh | bash
```

2 - install node

```
nvm install node
```

General steps

1 - install components

```
npm create vite@latest my-charting-site --template react
cd my-charting-site
npm install

npm install -D tailwindcss@3 postcss autoprefixer
npx tailwindcss init -p
```

2 - test

```
npm run dev
```

build

```
npm run build
```

Syncthing

Screen

Screen lets you make virtual terminals with

```
screen
```

New session with name

```
screen -S session_name
```

Tailscale

Client setup

1. Install Tailscale client

```
curl -fsSL https://tailscale.com/install.sh | sh
```

2. Create a pre-auth key on your Headscale server

```
docker exec headscale headscale preauthkeys create --user 1 --reusable --expiration 24h
```

Copy the key that's generated.

3. Connect to your Headscale server

```
sudo tailscale up --login-server=https://headscale.conorbriggs.com.au --authkey=YOUR_PREAUTH_KEY --hostname=rpi3
```

Replace YOUR_PREAUTH_KEY with the key from step 2.

4. Verify connection

```
# On Raspberry Pi:
sudo tailscale status

# On your Headscale server:
docker exec headscale headscale nodes list
```

You should see your Raspberry Pi listed with the hostname "raspberry-pi" and an IP like 100.64.0.2.

smb / samba

Samba Setup on Linux (Home Server)

A practical guide to installing and configuring Samba for a home network, covering a shared folder setup suitable for a home wiki or general file server.

---

1. Install Samba

```
sudo apt update && sudo apt install samba samba-common-bin -y
```

Verify it's running:

```
sudo systemctl status smbd nmbd
```

Both should be active. If not:

```
sudo systemctl enable smbd nmbd --now
```

---

2. Create the Share Directory

```
sudo mkdir -p /srv/samba/wiki
sudo chown -R conor:conor /srv/samba/wiki
sudo chmod 755 /srv/samba/wiki
```

Adjust owner to your actual Linux username.

---

3. Configure Samba

Back up the default config first:

```
sudo cp /etc/samba/smb.conf /etc/samba/smb.conf.bak
```

Edit the config:

```
sudo nano /etc/samba/smb.conf
```

Global section — replace or update:

```
[global]
   workgroup = WORKGROUP
   server string = Home Server
   server role = standalone server
   log file = /var/log/samba/log.%m
   max log size = 1000
   logging = file
   panic action = /usr/share/samba/panic-action %d

   # Security
   security = user
   map to guest = never
   encrypt passwords = yes

   # Performance
   socket options = TCP_NODELAY IPTOS_LOWDELAY
   read raw = yes
   write raw = yes
   use sendfile = yes
```

Share definition — add to the bottom:

```
[wiki]
   comment = Home Wiki
   path = /srv/samba/wiki
```

```
browseable = yes
read only = no
valid users = conor
create mask = 0664
directory mask = 0775
force group = conor
```

---

4. Create Samba User

Samba uses its own password store separate from Linux system passwords. The Linux user must already exist.

```
sudo smbpasswd -a conor
```

You'll be prompted to set a Samba-specific password. Enable the user:

```
sudo smbpasswd -e conor
```

---

5. Validate Config and Restart

Test the config for syntax errors:

```
testparm
```

If clean:

```
sudo systemctl restart smbd nmbd
```

---

6. Firewall

If UFW is active:

```
sudo ufw allow samba
```

This opens ports 137, 138 (UDP) and 139, 445 (TCP).

---

7. Connect From Clients

Linux (Files / Nautilus):

```
smb://SERVER_IP/wiki
```

Or mount via CLI:

```
sudo mount -t cifs //SERVER_IP/wiki /mnt/wiki -o username=conor,password=yourpass,uid=1000,gid=1000
```

Permanent via `/etc/fstab`:

```
//SERVER_IP/wiki  /mnt/wiki  cifs  credentials=/home/conor/.smbcredentials,uid=1000,gid=1000,_netdev,x-systemd.automount  0  0
```

`/home/conor/.smbcredentials`:

```
username=conor
password=yourpass
```

```
chmod 600 /home/conor/.smbcredentials
```

macOS:

Finder → Go → Connect to Server → `smb://SERVER_IP/wiki`

Windows:

`\\SERVER_IP\wiki` in Explorer address bar, or map as a network drive.

---

8. Tailscale Consideration

Since you're on Tailscale, use the Tailscale IP (`100.x.x.x`) instead of the LAN IP for consistent access across all your machines regardless of which network you're on. This also avoids exposing Samba to the public internet — Samba should **never** be exposed publicly, it has a long CVE history (EternalBlue, etc.).

Bind Samba only to your LAN + Tailscale interfaces to be safe:

```
[global]
   interfaces = lo tailscale0 eth0
   bind interfaces only = yes
```

Replace `eth0` with your actual LAN interface (`ip a` to check).

---

9. Verify the Share is Visible

From the server itself:

```
smbclient -L localhost -U conor
```

From another machine:

```
smbclient -L //SERVER_IP -U conor
```

You should see `wiki` listed under shares.

---

10. Troubleshooting

Symptom	Likely cause	Fix
"Permission denied" on connect	Wrong Samba password or user not enabled	`smbpasswd -e username`
Share not visible	`browseable = no` or firewall	Check config + `ufw status`
Can connect but can't write	Directory permissions	`chmod 775 /srv/samba/wiki`
Works on LAN, not Tailscale	Interfaces binding	Add `tailscale0` to `interfaces`
`testparm` errors	Config syntax	Read the output carefully, it's specific

---

Notes

- Samba passwords and Linux system passwords are **independent** — changing one doesn't change the other.
- For a multi-user setup, create a dedicated group (e.g., `sambashare`) and use `valid users = @sambashare` in the share definition.
- If you're running this on the same machine as your wiki app (e.g., WikiJS, Obsidian vault server), Samba is a good way to access the underlying vault files directly from other machines for editing.

autofs

autofs Setup and Configuration Guide

`autofs` mounts filesystems on-demand when accessed and unmounts them after a configurable idle timeout. This makes it significantly more robust than static `/etc/fstab` mounts for network shares — a missing or offline host won't hang your boot, and stale mounts are cleaned up automatically.

---

1. Install

```
sudo apt update && sudo apt install autofs sshfs -y
```

For NFS shares also install:

```
sudo apt install nfs-common -y
```

---

2. How autofs Works

autofs has two config layers:

File	Purpose
`/etc/auto.master`	Master map — defines mount point directories and which map file handles them
`/etc/auto.<name>`	Detail map — defines individual mounts under that directory

When you access `/mnt/remote/myshare`, autofs intercepts it, reads the map, and mounts the share on the fly. After the timeout with no activity, it unmounts automatically.

3. Master Map — `/etc/auto.master`

Each line in `auto.master` follows this format:

```
<base_mount_dir>  <map_file>  [options]
```

Example:

```
/mnt/remote  /etc/auto.sshfs  --timeout=60,--ghost
/mnt/nfs     /etc/auto.nfs    --timeout=120,--ghost
```

Key options:

Option	Effect
`--timeout=N`	Unmount after N seconds of inactivity
`--ghost`	Creates empty placeholder directories so the mount point is visible even when unmounted. Without this, `ls /mnt/remote` shows nothing until a mount is active
`--verbose`	Useful during setup/debugging

Important: auto.master entry placement

The default `/etc/auto.master` contains a `+auto.master` line which is a legacy NIS include directive. Your custom entries must go **above** this line, otherwise they may be ignored on some systems. The `+auto.master` line is safe to remove entirely on a standalone home server with no NIS infrastructure.

Clean minimal auto.master:

```
+dir:/etc/auto.master.d


/mnt/remote  /etc/auto.sshfs  --timeout=60,--ghost
```

The `+dir:/etc/auto.master.d` line is worth keeping — it allows dropping extra `.autofs` map files into that directory without editing auto.master directly.

4. Detail Map Files

SSHFS (SSH/SFTP) — Modern Syntax

> **Important:** Many guides online show legacy syntax using `fstype=fuse` and `:sshfs#user@host` format. This does **not** work on modern Linux kernels. Always use `fstype=fuse.sshfs` with plain `user@host:/path` syntax.

Create `/etc/auto.sshfs`:

```
<mount_name>  -fstype=fuse.sshfs,rw,allow_other,IdentityFile=/home/conor/.ssh/id_ed25519,uid=1000,gid=1000,StrictHostKeyChecking=no,ServerAliveInterval=15,ServerAliveCountMax=3  conor@100.x.x.x:/home/conor
```

Breakdown:

Part	Meaning
`<mount_name>`	The subdirectory created under the base — e.g. `farmdesktop` becomes `/mnt/remote/farmdesktop`
`-fstype=fuse.sshfs`	Correct modern fstype for SSHFS mounts
`rw`	Read/write
`allow_other`	Allows users other than root to access the mount
`IdentityFile=`	Path to SSH private key — **must use key auth, not password**
`uid=1000,gid=1000`	Map remote files to your local user. Check yours with `id conor`
`StrictHostKeyChecking=no`	Required because autofs runs as root which has an empty known_hosts — without this the mount hangs waiting for an interactive host key confirmation that never comes
`ServerAliveInterval=15`	Send keepalive every 15s to prevent silent disconnects
`ServerAliveCountMax=3`	Drop connection after 3 missed keepalives

Full working example:

```
farm-server  -fstype=fuse.sshfs,rw,allow_other,IdentityFile=/home/conor/.ssh/cbcore-key,uid=1000,gid=1000,StrictHostKeyChecking=no,ServerAliveInterval=15,ServerAliveCountMax=3 conor@100.64.0.12:/home/conor
```

Access via: `/mnt/remote/farm-server`

Why legacy syntax fails

Old guides show:

```
farmdesktop  -fstype=fuse,rw,allow_other,...  :sshfs#conor@100.64.0.5:/home/conor
```

This tells autofs to call `mount -t fuse` with `sshfs#` as the device — an approach that relied on the generic FUSE kernel module handling the mount. Modern kernels expect `mount -t fuse.sshfs` with a plain remote path. Using the old syntax results in `signal 22 (SIGABRT)` and a failed mount with no useful error message.

NFS

Create `/etc/auto.nfs`:

```
nas  -fstype=nfs4,rw,soft,intr  100.64.0.10:/exports/data
```

Option	Meaning
`nfs4`	Use NFSv4 (preferred)
`soft`	Return error on timeout instead of hanging indefinitely
`intr`	Allow interrupting a hung NFS call with Ctrl+C

Samba / CIFS

Create `/etc/auto.smb`:

```
wiki  -fstype=cifs,rw,credentials=/home/conor/.smbcredentials,uid=1000,gid=1000,iocharset=utf8
://100.64.0.10/wiki
```

`/home/conor/.smbcredentials`:

```
username=conor
password=yourpassword
```

```
chmod 600 /home/conor/.smbcredentials
```

5. FUSE Prerequisite for SSHFS

`allow_other` requires this line to be uncommented in `/etc/fuse.conf`:

```
sudo nano /etc/fuse.conf
```

Uncomment:

```
user_allow_other
```

Without this, SSHFS mounts via autofs (which runs as root) will be inaccessible to your user account.

---

6. SSH Key Setup

autofs runs as root, so two things must be true:

1. The key file must be readable by root
2. Root must have the remote host in its known_hosts — **or** `StrictHostKeyChecking=no` must be set

Seed root's known_hosts before first use:

```
sudo ssh -i /home/conor/.ssh/your-key conor@100.x.x.x echo "ok"
# Type "yes" when prompted to accept the host key
```

Alternatively, use `StrictHostKeyChecking=no` in the map file options (acceptable on a private Tailscale network).

Option A — use your existing user key (works if permissions allow root to read it):

```
IdentityFile=/home/conor/.ssh/id_ed25519
```

Option B — create a dedicated root-owned key for autofs:

```
sudo ssh-keygen -t ed25519 -f /root/.ssh/autofs_id_ed25519 -N ""
sudo ssh-copy-id -i /root/.ssh/autofs_id_ed25519.pub conor@100.x.x.x
```

Then use `IdentityFile=/root/.ssh/autofs_id_ed25519` in the map file. Cleaner — keeps autofs auth separate from your interactive SSH key.

---

7. Enable and Start autofs

```
sudo systemctl enable autofs --now
```

Reload after map file changes:

```
sudo systemctl reload autofs
```

Full restart required after auto.master changes:

```
sudo systemctl restart autofs
```

---

8. Testing

Trigger a mount by accessing the path directly:

```
ls /mnt/remote/farm-server
```

Check what's currently mounted:

```
mount | grep autofs
```

Verify autofs has loaded your maps correctly:

```
sudo automount --dumpmaps
```

Check status and logs:

```
sudo systemctl status autofs
sudo journalctl -u autofs -f
```

Force unmount manually:

```
sudo umount /mnt/remote/farm-server
```

9. Debugging a Failed Mount

If the mount silently fails, run autofs in foreground debug mode:

```
sudo systemctl stop autofs
sudo automount -f --verbose --debug 2>&1 | tee /tmp/autofs-debug.log &
sleep 2
ls /mnt/remote/farm-server
sleep 2
cat /tmp/autofs-debug.log
```

Then kill the debug instance and restart the service:

```
sudo pkill -9 -f automount
sudo rm -f /run/autofs.pid
sudo systemctl start autofs
```

If the mount command itself hangs, test it manually first:

```
# Test SSH connectivity as root
sudo ssh -i /path/to/key user@host echo "ok"

# Test sshfs as your normal user
mkdir /tmp/testmount
sshfs -o IdentityFile=/path/to/key user@host:/remote/path /tmp/testmount
ls /tmp/testmount
```

If sshfs works as your user but hangs as root, the issue is root's SSH environment (known_hosts, key permissions). Add `StrictHostKeyChecking=no` to the map options and ensure the key is readable by root.

---

10. Troubleshooting

Symptom	Likely cause	Fix

`ls` on mount point returns "No such file or directory"	autofs not reading map, or entry placement in auto.master	Run `sudo automount --dumpmaps` to verify map is loaded; check entry is above `+auto.master`
Mount hangs on access	Root's known_hosts empty — SSH waiting for interactive host key confirmation	Add `StrictHostKeyChecking=no` to map options, or pre-seed with `sudo ssh user@host echo ok`
`signal 22` / SIGABRT in debug output	Legacy `fstype=fuse` + `sshfs#` syntax	Change to `fstype=fuse.sshfs` and plain `user@host:/path`
"Permission denied" accessing mounted path	`allow_other` not set or `user_allow_other` not in fuse.conf	Check `/etc/fuse.conf`
Mount works as root but not as user	`uid=` / `gid=` not set in map file	Add `uid=1000,gid=1000` (check with `id username`)
autofs fails to start after manual debug session	Stale PID file from debug automount process	`sudo pkill -9 -f automount && sudo rm -f /run/autofs.pid`
Changes to map file have no effect	autofs cached old config	`sudo systemctl reload autofs`
`ls /mnt/remote` shows nothing despite ghost mode	Ghost mode requires autofs to be fully running and map parsed	Run `sudo automount --dumpmaps` to confirm map loaded

---

11. Full Example: Tailscale Homelab

`/etc/auto.master`:

```
+dir:/etc/auto.master.d

/mnt/remote  /etc/auto.sshfs  --timeout=60,--ghost
```

`/etc/auto.sshfs`:

```
farm-pi      -
fstype=fuse.sshfs,rw,allow_other,IdentityFile=/root/.ssh/autofs_id_ed25519,uid=1000,gid=1000,StrictHostKeyCh
ecking=no,ServerAliveInterval=15,ServerAliveCountMax=3  conor@100.64.0.2:/home/conor
farmdesktop  -
fstype=fuse.sshfs,rw,allow_other,IdentityFile=/root/.ssh/autofs_id_ed25519,uid=1000,gid=1000,StrictHostKeyCh
ecking=no,ServerAliveInterval=15,ServerAliveCountMax=3  conor@100.64.0.3:/home/conor
homeserver   -
fstype=fuse.sshfs,rw,allow_other,IdentityFile=/root/.ssh/autofs_id_ed25519,uid=1000,gid=1000,StrictHostKeyCh
```

```
ecking=no,ServerAliveInterval=15,ServerAliveCountMax=3  conor@100.64.0.4:/home/conor
```

Access:

```
/mnt/remote/farm-pi/
/mnt/remote/farmdesktop/
/mnt/remote/homeserver/
```

All mount on first access, unmount after 60 seconds idle. If a farm machine is offline, accessing its path returns an error immediately rather than hanging the system.

Notes

- autofs mounts do **not** appear in `df` or `mount` output unless currently active — this is normal.
- `/etc/auto.master` changes require a full `systemctl restart autofs`; map file changes only need `systemctl reload autofs`.
- For Ansible management across your fleet, the map files and master config are straightforward to template — a single playbook can deploy consistent autofs config to all five machines.