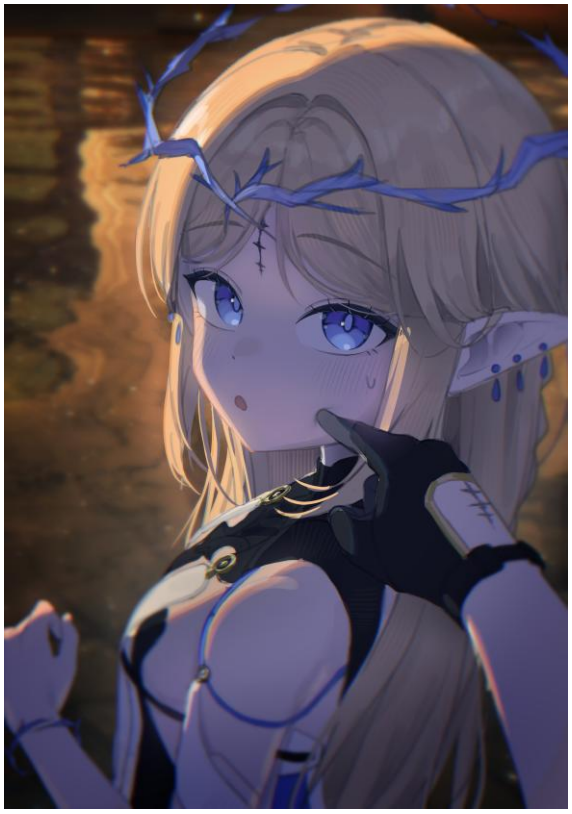


ARCH LINUX INSTALLATION GUIDE

Without Arch Installer | Newbie Edition

ABSTRACT

This guide walks a first-time Linux user through a manual Arch Linux installation (UEFI/GPT) without the graphical installer. You'll create a bootable USB, partition and format disks, mount and generate fstab, and install a minimal base system with pacstrap. Then we configure timezone, locale, users and sudo, networking, and set up GRUB to boot. After first boot we add a friendly desktop (KDE Plasma), essential tools, and AUR support with yay, plus some quality-of-life checks and safe cleanup commands. Optional chapters introduce networking hardening (DoH + firewall), Secure Boot, virtualization/containers, Wine, and ricing idea(s).

Goal: a clean, working Arch install you understand well enough to maintain and customize.

Rhio Bimo | 13523123

"One shall reclaim thy V-card after properly reading and completing the installation for Arch Linux"

- Some Arch user, btw

Preface

We're going to be using the official documentation for Arch Linux installation guide from the official website to install, you can view it ***here***. And why 'Newbie Edition' you might ask. Well, cause this is also my first time installing any Linux Distros and I chose Arch as my first one :v.

This guide will be using a USB stick (drive) to install the Linux system into your machine. So, if you already have a USB Drive/Stick that's formatted to run in the *firmware boot* that has the Arch iso, then you can skip to ***Configuring the System***.

A little disclaimer, this document **will only be covering UEFI install**. Which means, if you have a machine that's really old, this document might not be compatible. If you're in doubt and you're using window right now, just run **'msinfo32'** and see if your bios uses UEFI, if you're on Linux, you can run efibootmgr and if it returns a boot list, you're probably in UEFI mode. If you currently don't have access to any operating system, google it, idk.

Take note that this guide **will be wiping all your files from your machine**, so if you have important files, please back them up separately.

Here's the things that we'll be doing in this guide:

No.	Features
1.	Creating and Formatting USB Stick
2.	Connecting to WiFi
3.	Localization
4.	Adding User (with sudo privilege)
5.	GUI, Sounds, Tiling Window Manager
6.	Installing Packages (VSCodium, Web Browser, Wine)
7.	Custom Bootloader
8.	Installing LINE and STEAM (using wine)
9.	~~Enabling Secure Booting~~
10.	Wayland
11.	Anime from Konsole
12.	Doom in Terminal
13.	Playing Non-Linux Supported Game
14.	Virtual Machine and Container
15.	Ricing

Table of Contents

I. Formatting USB Stick

1.1. Installing Prerequisites

- Install **qBittorrent** for installing the Arch Linux's Image
- Install **rufus** on Windows or **etcher** on Unix machine (Linux/MAC)

You can then install the program needed by just a quick online search and you'd be done with this step.

1.2. Installing ISO

Make sure to have already installed QBittorent. You can install the Arch's ISO from the website then clicking "**Magnet link for 2025.08.01**" (You can try clicking the one in this document too, but be careful). Then you should be prompted to (or even automatically) open your torrent program. Wait for it until it has finished downloading.

When finished (usually it'll say 'seeding') make sure the .iso file is in your downloads folder and if it is, you're good to go! Just close the torrent program (and if to also make sure it's not running in the background) and you can start formatting the disk!

1.3. Formatting the Disk

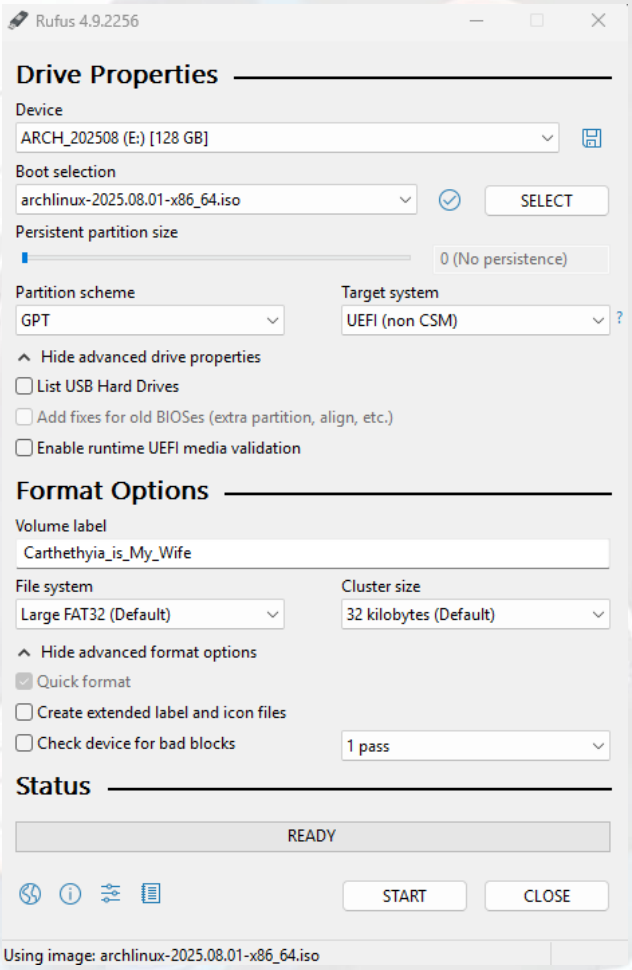
First, make sure your **USB is clean**, because **every data will be destroyed**. If you have any important data on your disk, be sure to **back them up first** and go back into this section.

Once done, you can then plug the USB stick to your machine.

1.3.1. For Windows

For windows, I'll be using rufus. You can just run the already installed rufus.exe file and the following should pop up.

Side note, if you think the following is hard to follow, you can also follow the formatting for Unix down below. But rufus will give you a more customizable formatting option.



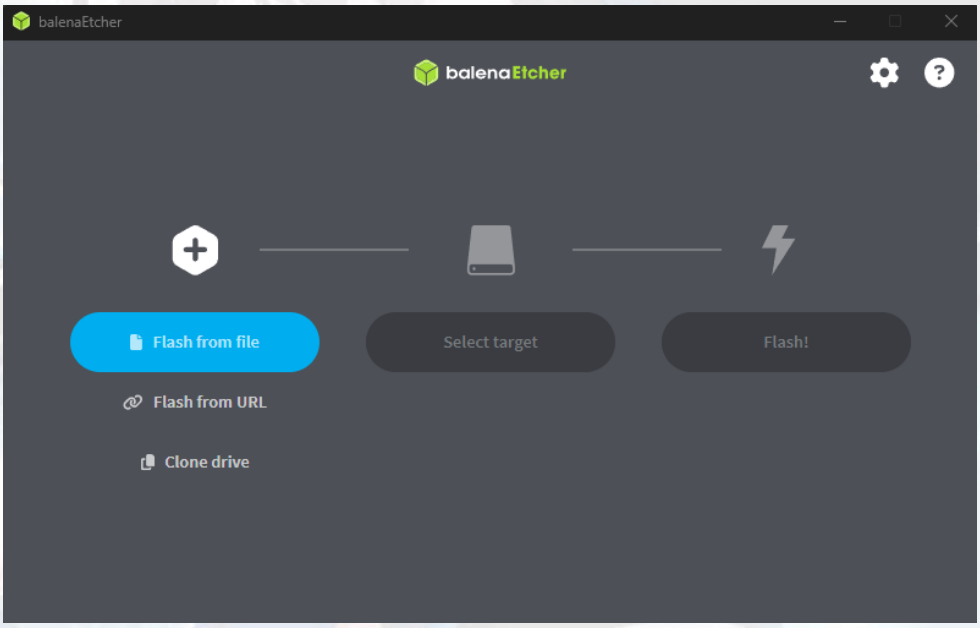
- Device: Your USB Disk
- Boot Selection: Click ‘SELECT’ and choose your downloaded archlinux-XXX.iso image
- Persistent partition size: Just 0, it doesn’t really work with Arch anyway
- Partition sheme: GPT
- Target system: UEFI
- Advance drive properties: just turn all of it off
- Volume label: “Carthethyia_is_MY_WIFE” (optional). This will be your disk’s name. Take note that some string format is not allowed, as well as non-printable characters
- File system: FAT32 (The image shows “Large FAT32” because I have a lot of space in the disk, but usually {and if you have the choice} just use normal FAT32)
- Advance format options: Make sure quick format is turned on, the rest you can turn it off.

Once you're done setting things up, just click start, get out of your chair, and brew some tea while we wait!

1.3.2. For Linux/MAC

For Unix system, we'll be using etcher. This is quite simple as in you just need to launch etcher. Though a little disclaimer, if you want a more customizable option, you can use [Ventoy](), but for this tutorial purpose's we'll use etcher.

The etcher's (balenaEtcher) Ui should look like the following:



For this, just select 'Flash from file' your archlinux_XXX.iso file, select your target USB stick, and click Flash! You can then wait, brew a few coffees, and wait until it's done!

Once formatting the disk(s) is done, then you're ready to go into

1.4. Go Into Boot Menu

Now, plug your USB stick and go into Boot Manager. In most device, going into boot manager usually uses the F12 key, pressing it while the system is booting. But some dive might be different, so I'll list it below:

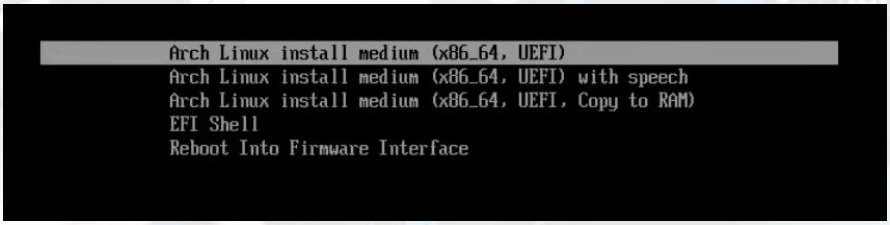
Manufacturer	Typical Boot Menu Key	Typical Firmware Settings Key
(Default)	F12	F2
Acer	F12	Del / F2
ASUS	Esc / F8	Del / F2
Dell	F12	F2

HP	Esc (then F9 for Boot Menu)	Esc (then F10 for BIOS)
Lenovo	F12 / Novo Button	F1 / F2 / Novo Button
MSI	F11	Del
Toshiba	F12	F2
Samsung	Esc / F12	F2
Sony	F11	F2

If none of your device are listed here, you can just search it online.

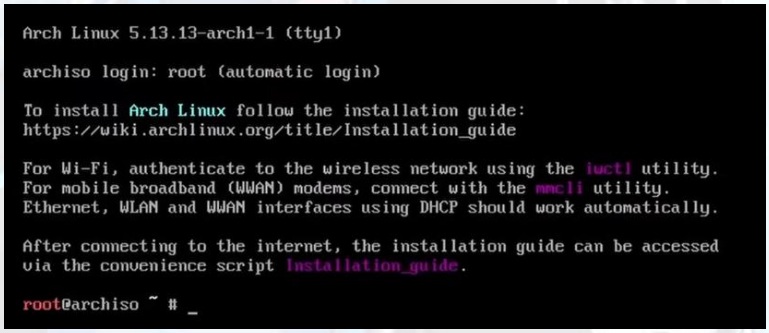
After doing that, you would see many different devices. If you plug your USB in, you should be able to see your USB's name within the boot menu, go towards it (using the arrow keys) and select it (using 'Enter/Return' key).

If you have done things properly, I just want to say that **some devices might start to sound like they're about to EXPLODE and THAT'S OKAY!** You can take a moment to make sure that the screen you're seeing is like this:



If your screen looks like that, then you can **either** let the timer go down to zero and **don't touch anything**, or just press the 'Enter/Return' and **select the first** options.

Let it boot, when done, your screen should look something like this:



The first thing you probably want to set is your keyboard layout. The default keyboard layout should be US, so if you don't want to set it you can skip towards the next part.

This is how to set your keyboard layout:

```
# loadkeys {YOUR_LANGUAGE} #You don't have to put '#' when typing
# for example:
# Loadkeys jp # <-- This will use Japanese keys layout
```

1.5. Connecting to Internet (IWD)

This is a pretty important step, so be sure to follow closely. You can also follow the guide in the wiki page for this step, but nonetheless here's the basic steps:

1. Check if you're connected to the internet by pinging

```
# ping -c3 google.com
```

If the response from the command is not an immediate error and it prints our something along the lines of "--- google.com ping statistics ---" the you're good! You can skip to partitioning the disk.

If you get no response/error message, then continue.

Fun Fact! If you run the ping command without the '-c3' words (and that is called a "flag") it will run the ping command indefinitely! To stop any commands that's running in, you can press 'ctrl+c' which will terminate the current running program! Try it out!

2. Run IWD

You can start running IWD by doing:

```
# iwctl
```

That command should bring you into iwd's interactive prompt. Your bash should now look along the line of:

```
[iwd]#
```

If that doesn't work (it doesn't bring you into iwd's interactive prompt) then you probably need to set an ethernet connection into your machine.

3. List the devices available

```
[iwd]# device list
```

You should see the list of available devices for connecting into the internet (i.e. wlan0, wlsp0, etc.) and adapter type (i.e. phy0). Make sure to remember that as NAME, as well as the adapter as ADAPTER for the next steps.

If you don't see any available device, then your machine needs to be equipped with one. For this, you can either connect your machine to with ethernet (which also should let you skip this entire process and go to the next chapter) or you could buy a WiFi adapter for PC or laptops, they kinda look like this.

Before that, you should also see whether or not your device is turned on/off. If it's on, then you're good! If it's off, turn it on by:

```
[iwd]# device NAME set-property Powered on
[iwd]# adapter ADAPTER set-property Powered on
```

You should then list the device again to see if it has turned on (which it should)

4. See available network

```
[iwd]# station NAME scan
[iwd]# station NAME get-networks
```

That should output a list of available network names. In networking, a network's (WiFi) name is usually called an SSID.

5. Connect to network

```
[iwd]# station NAME connect SSID
# if the network needs a password, you'll also be prompted to input its password
```

If you know that your network SHOULD be there, it probably means the network is hidden. To connect to hidden network, do this:

```
[iwd]# station NAME connect-hidden SSID
# if the network needs a password, you'll also be prompted to input its password
```

It should output that the network is connected or if your password input is incorrect.

6. Final check

Finally, we should check whether or not we're now connected to the internet! First we must exit IWD.

```
[iwd]# exit
```

You should then see screen go back to 'root@archiso'. You can then do the same ping as the one showed in the first part:

```
# ping -c3 google.com
```

And your output should be around like this:

```
root@archiso ~ # ping -c3 google.com
PING forcesafesearch.google.com (216.239.38.120) 56(84) bytes of data.
64 bytes from any-in-2678.1e100.net (216.239.38.120): icmp_seq=1 ttl=255 time=20.2 ms
64 bytes from any-in-2678.1e100.net (216.239.38.120): icmp_seq=2 ttl=255 time=20.0 ms
64 bytes from any-in-2678.1e100.net (216.239.38.120): icmp_seq=3 ttl=255 time=21.0 ms

--- forcesafesearch.google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2879ms
rtt min/avg/max/mdev = 19.973/20.391/21.000/0.440 ms
```

If it is, then you're good! Let's go to the next step!

Oh yeah! Your screen now is probably cluttered with a lot of mess, to clean it up you can either type 'clear' and press the 'Enter/Return' key or press ctrl+L to clear the screen!

II. Configuring the System

I'm assuming that you have already have the USB stick and went into the Installation for Arch and have connected to the internet. If you haven't went inside the installation interface, click here. If you're using an ethernet, then you're good, but if you're using WLAN then please take a step back and check if you're connected to the internet (you can do this by pinging). If not, please follow this guide.

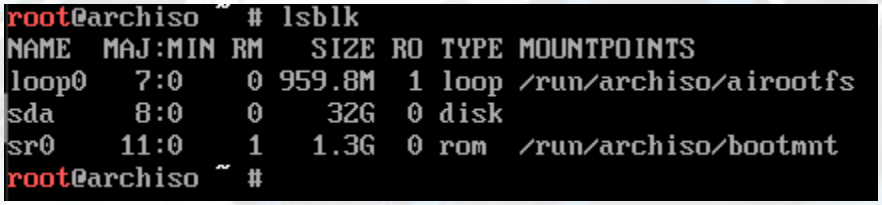
When everything's in check, then ur good! You can continue below.

2.1. Partitioning the Disk

If you're installing linux into your main machine (not VM), you might want to do:

```
# lsblk
```

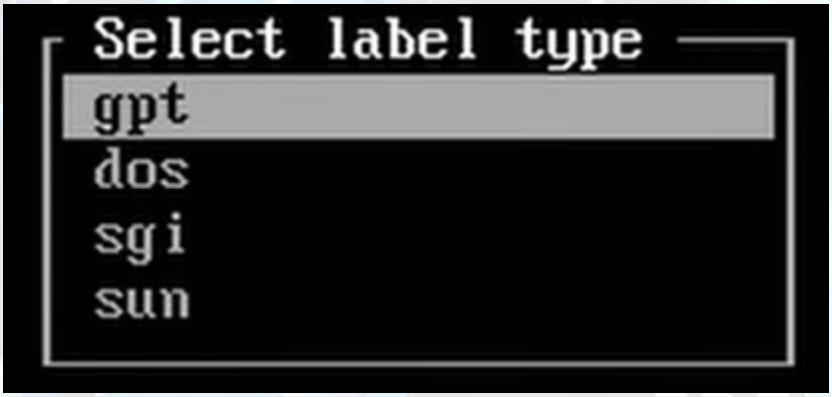
In which it will show a screen like this:



Make sure you know what type of storage (SSD, NVMe, HDD, etc.) that your current machine actually uses. You will then use this command:

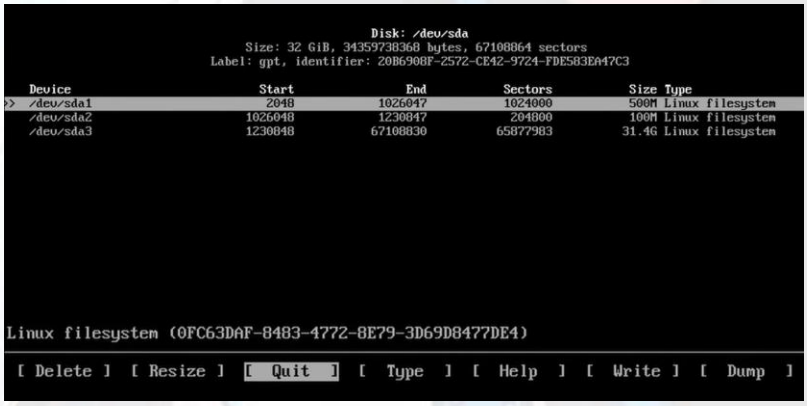
```
# cfdisk {YOUR_DISK_NAME}
# example:
# cfdisk nvme0n1
```

You may then encounter a screen like the following

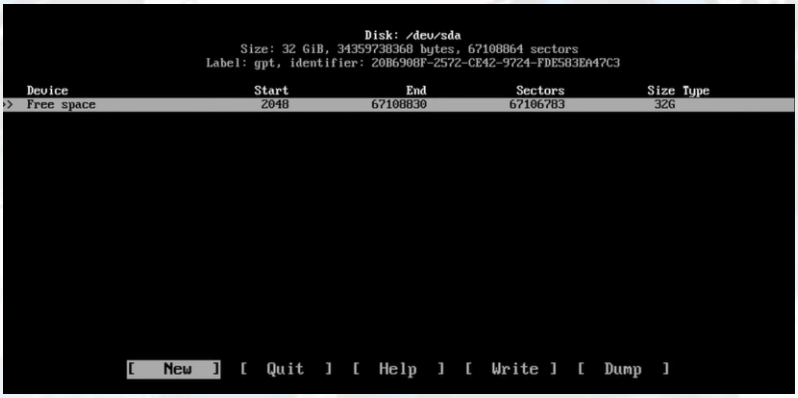


If you didn't encounter it, it's also fine! For this, you want to select 'gpt' (by using your arrow keys) and confirm it (by using the 'Enter/Return' key).

Your screen will then look like this:



If you're in a VM or a new Machine, it may only show a single "Free Space" like this:



And that's okay! And you're good for now (can skip the first point 'Delete other partition').
If you encounter something similar like the first screen, the you must first do these:

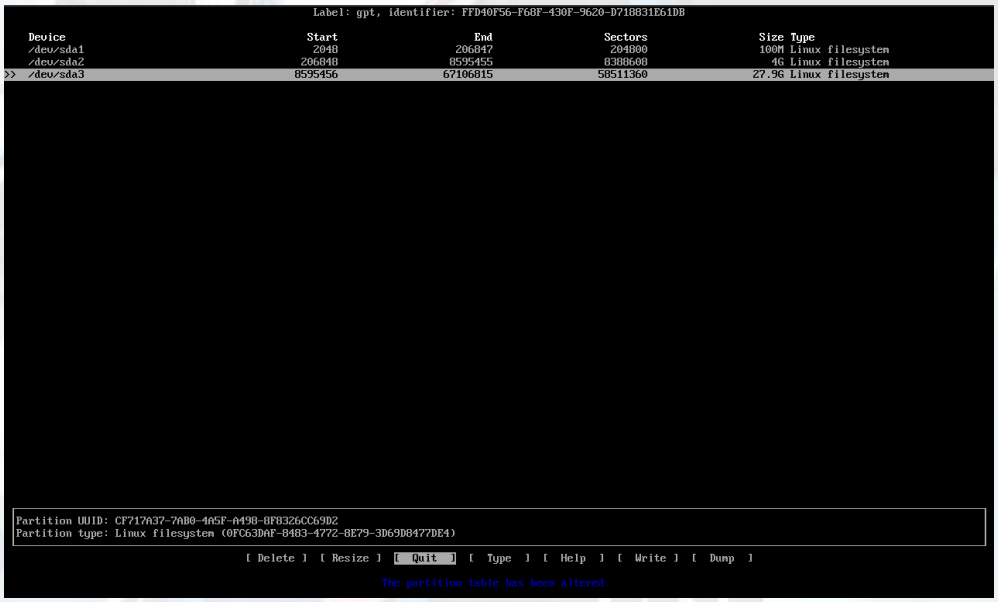
1. Delete other partition
    a. Selecting the partition: using up and down keys until your
    b. Deleting it: using arrow left and right arrow keys to navigate towards 'delete' in the menu below and pressing the 'Enter' keys.
    c. Do all of that until you're left with the something like the second image, above.
2. Partitioning:
    a. Select the 'New' command (use the left and right arrow key) and press the 'Enter/Return' key.
    b. You will then have to write your 'Partition size', for this one you would want to type '**100M**' and that will be our **boot partition**, you can the press 'Enter/Return'

c. Navigate towards the ‘Free space’ and add another new partition, this time you will write ‘**4G**’ at least and if your machine has enough storage, you can go up to ‘**16G**’. This will be our **swap partition**, when you’re set just press ‘Enter/Return’

d. The third partition will be filled with however many disks memory you have left. So just navigate towards ‘Free space’, select ‘New’, and just press ‘Enter/Return’ twice. This will be our **root partition**.

e. Lastly, you will have to write it into the disk, so just navigate towards the ‘Write’ option and press ‘Enter/Return’. You will then be prompted to type ‘**yes**’, so just type ‘yes’ nad press ‘Enter/Return’ again.

f. After doing all of that, your screen should look like this:

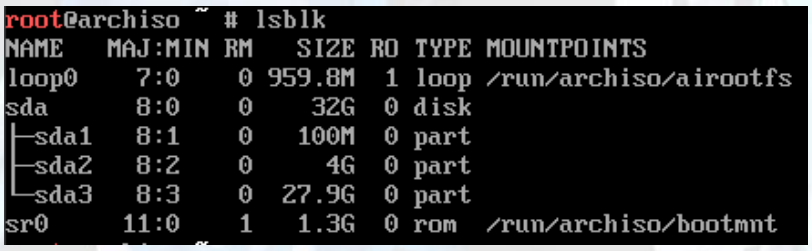


If it is, then you’re good! Just navigate towards the ‘Qui’ option and press ‘Enter/Return’ to quit cfdisk. You can the clear the screen by pressing ‘ctrl+L’.

One final check, do:

```
# lsblk
```

And depending on what disk your partition, it will look a little different, but it should look something like this:



From the image shown above, take note that:

- sda1 : Boot Partition (100M size)
- sda2 : Swap Partition (4/8/16G size)
- sda3 : Root Partition (Should be the biggest out of the three)

2.2. Formatting the Partition

2.2.1. Formatting Root Partition

```
# mkfs.ext4 /dev/{YOUR_DISK_PARTITION}
# example:
# mkfs.ext4 /dev/sda3
```

After that, just press 'Enter/Return'

2.2.2. Formatting Boot Partition

```
# mkfs.fat -F 32 /dev/{YOUR_DISK_PARTITION}
# example:
# mkfs.fat -F 32 /dev/sda1
```

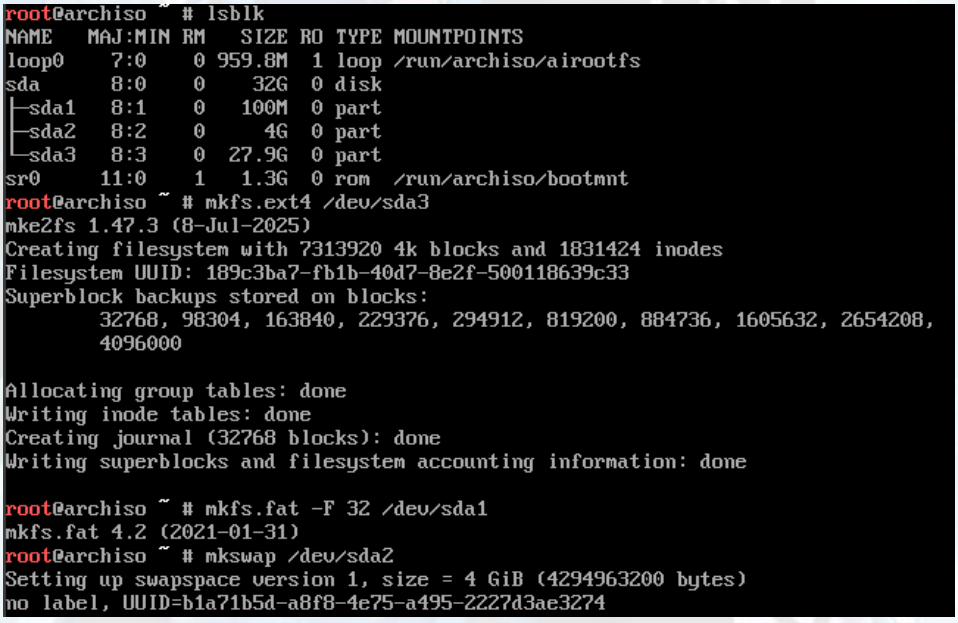
After that, just press 'Enter/Return'

2.2.3. Swap Partition

```
# mkswap /dev/{YOUR_DISK_PARTITION}
# example:
# mkswap /dev/sda2
```

After that, just press 'Enter/Return'

This image shows an example on what it will look like in your screen:



2.3. Mounting the Partition

2.3.1. Mounting Root Partition

```
# mount /dev/{YOUR_DISK_PARTITION} /mnt
# example:
#  mount /dev/sda3 /mnt
```

After that, just press 'Enter/Return'

2.3.2. Mounting Boot Partition

The boot partition should be mounted in 'boot/efi' so we should make the directory first by doing:

```
# mkdir -p /mnt/boot/efi
```

Press 'Enter/Return' and after that do:

```
# mount /dev/{YOUR_DISK_PARTITION} /mnt/boot/efi
# example:
# mount /dev/sda1 /mnt/boot/efi
```

After that, just press 'Enter/Return'

2.3.3. Turning on Swap Partition

```
# swapon /dev/{YOUR_DISK_PARTITION}
# example:
# swapon /dev/sda2
```

After that, just press 'Enter/Return'

Finally, we're going to check our blocks by doing:

```
# lsblk
```

After doing that, your screen will look a little something like this:

```
root@archiso ~ # mount /dev/sda3 /mnt
root@archiso ~ # mkdir -p /mnt/boot/efi
root@archiso ~ # mount /dev/sda1 /mnt/boot/efi
root@archiso ~ # swapon /dev/sda2
root@archiso ~ # lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
loop0    7:0    0 959.8M  1 loop /run/archiso/airootfs
sda      8:0    0    32G  0 disk
├─sda1   8:1    0   100M  0 part /mnt/boot/efi
├─sda2   8:2    0     4G  0 part [SWAP]
└─sda3   8:3    0  27.9G  0 part /mnt
                                 /mnt
sr0     11:0    1   1.3G  0 rom  /run/archiso/bootmnt
```

You can see that sda1's mountpoints are in '/mnt/boot/efi', sda2 is '[SWAP]', and sda3 is '/mnt'. If your screen looks like that, the you're good to go!

2.4. Installing Base System

If you want, just make sure that you're once again connected to the internet by pinging. If you're sure about that, you can then continue installing the base system the 'fun' part.

Before running the code below, let me tell you what we're installing right now is only the first few necessary packages, so just follow it for now because later on, you can freely install any packages you want.

The first few packages we're going to install are:

- base linux linux-firmware : This is basically the base Linux OS
- sof-firmware : Sounds for newer soundcard
- base-devel : For installing more package down the line
- grub : For actually booting into Arch Linux
- efibootmgr : For EFI support in grub
- nano : For writing script (essential for this tutorial)
- networkmanager : For connecting to network when Arch Linux is up
- amd-ucode* : For if you're using **amd cpu**, security updates
- intel-ucode* : For if you're using **intel cpu**, security updates

*) When installing this, I'm using a VM, so I don't need to do this, but if you're installing it into a machine, please look at your machine's CPU and adjust accordingly

Your command should look like this:

```
# pacstrap /mnt base linux linux-firmware sof-firmware\
base-devel grub efibootmgr nano networkmanager {CPU-ucode}
```

After making sure you don't have any typos, you can press 'Enter/Return', make sure the download is running properly (no error message), if the program prompts you to press 'Enter/Return' again, then just do what it prompts you. Other than that, just take another walk outside, brew another cup of tea and wait it out.

When the program has kicked you back into 'root@archiso' and you see no error message(s), then you're good to go for the next step!

2.5. Fstab

For this step, we'll be generating the Filesystem Tab (Fstab). Btw, if you haven't cleared your terminal yet, go ahead and clear it using 'ctrl+L'.

We first need to check if our partition is mounted correctly by doing:

```
# genfstab /mnt
```

Your screen will show you something along the line of this:

Here you can once again make sure that your partition is mounted to the correct mountpoint, as well as its format.

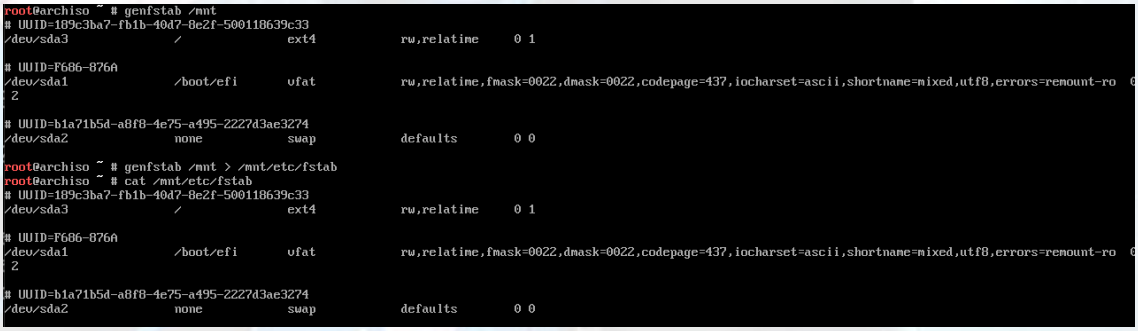
Once sure, do:

```
# genfstab /mnt > /mnt/etc/fstab
```

The '>' will then sends the output of 'genfstab /mnt' into '/mnt/etc/fstab'. Once you do that, make sure that the content inside '/mnt/etc/fstab' is the same as like running 'genfstab /mnt' by doing:

```
# cat /mnt/etc/fstab
```

Your screen will look something like this:

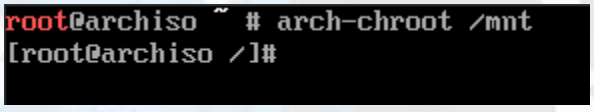


If your screen looks like that, then you're good to go!

Now you can finally go into your filesystem by doing:

```
# arch-chroot /mnt
```

This will change your arch root into your file system, and your 'root' in the screen should now not be red, and everything is just black and white. You screen will look like this:



2.6. Timezone and Localization

We first need to set our Timezone and Localization, this is important for setting up your device's language and stuff like that. A little disclaimer, if you're someone that uses a non-latin script like Japanese, Chinese, Thai, etc. I suggest to stick with the US localization for now, because the system does not support non-latin character yet (you need to install it later).

This is a pretty confusing step, so pay close attention!

```
# ln -sf /usr/share/zoneinfo/{ZONE}/{SUBZONE}\
/etc/localtime
# list of example:
# ln -sf /usr/share/zoneinfo/Asia/Tokyo /etc/localtime
# ln -sf /usr/share/zoneinfo/America/New_York \
 /etc/localtime
# ln -sf /usr/share/zoneinfo/Europe/London /etc/localtime
```

A little tip in finding your ZONE and SUBZONE is that you can go to this Wikipedia link. You can also press the 'Tab' on your keyboard to autocomplete the ZONE and SUBZONE if you have already put in the first few characters, if it doesn't autocomplete, then what you're typing is either too vague or doesn't exist in the list.

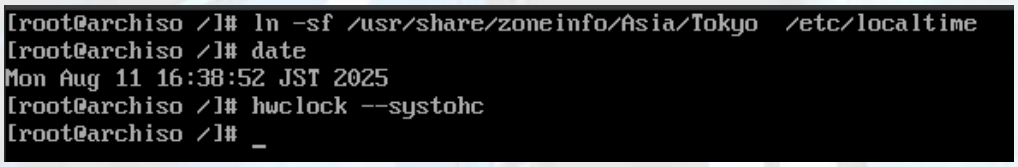
After doing that, you can then do:

```
# date
```

To check whether or not the output is correct with you selected time zone. Next just do

```
# hwclock --systohc
```

Which will synchronize the system clock. If you're not sure if you're doing things correctly, here's my screen setup.

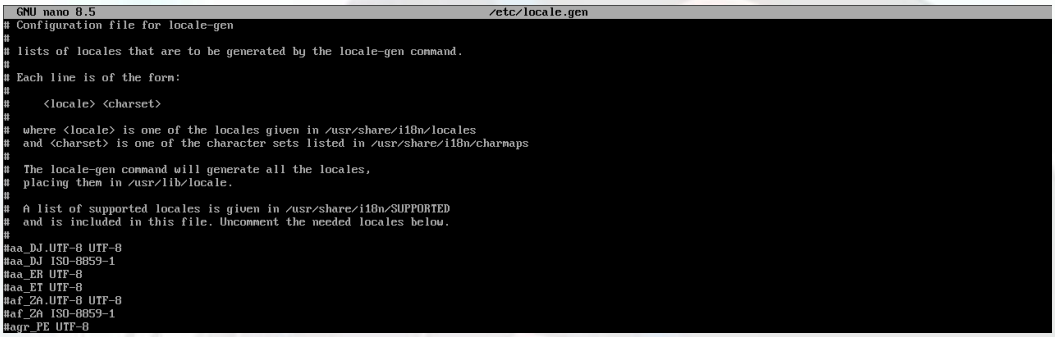


Next, we'll setup localization. Here's the steps:

1. Opening /etc/locale.gen

   ```
   # nano /etc/locale.gen
   ```

   When running that command, your screen will look something like this:



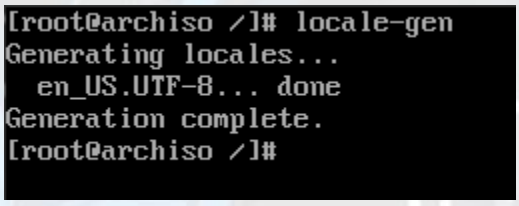
In here, you can use your arrow keys to navigate within the interface. Use your up and down arrow keys to edit different lines of code and scroll down the document, and left and right key to specify what letter you're editing.

In here, you want to find your desired locale, for this demonstration, I'll be looking for '**#en_US.UTF-8 UTF-8**'. After finding it, just **delete the leading '#'** and make it something like '**en_US.UTF-8 UTF-8**'. After doing that, make sure to write(save) it by pressing 'ctrl+O' (write) and then 'Enter/Return' (confirm changes) and you can exit the interface by pressing 'ctrl+X' (exit) and then 'ctrl+L' (clear).

2. Generating locale

```
# locale-gen
```

You are now generating the locale, once done, it should look something like this:



3. Specifying our locale

```
# nano /etc/locale.conf
```

Take note that some people may already have something in this file, and it should be fine, just make sure that the inside is like the following:

If you don't have anything inside of that yet, then just type the locale that you got printed in step 2 (the "Generating locales… **en_US.UTF-8**" <-- The bolded letter) into something like this "LANG={LOCALE}".

After doing that, make sure to write(save) it by pressing 'ctrl+O' (write) and then 'Enter/Return' (confirm changes) and you can exit the interface by pressing 'ctrl+X' (exit) and then 'ctrl+L' (clear).

2.7. Hostname and Adding Users

To specify the hostname (machine name) we'll do:

```
# nano /etc/hostname
```

Then you will be brought into GNU nano. You can then just type your host name anything you want. For me, I'll be naming it 'Arch-chan'. Don't forget to write (save) it before exiting GNU nano.

We'll now set up our root password. 'root' is basically a user that has all the privileges in modifying your OS, so setting up a password for it is very essential. Here's how:

```
# passwd
Then here you will be prompted to put your password
Here you will be prompted to confirm it
```

This is what it looks like in my screen. I will also suggest you write down your password or remember it very dearly. Because if you forgot about it, you're fucked.

```
[root@archiso /]# passwd
New password:
Retype new password:
passwd: password updated successfully
[root@archiso /]# _
```

We'll now also create a new user. This will be necessary for extra security and it'll just look nice 😊

```
# useradd -m -G wheel -s /bin/bash {YOUR_USER_NAME}
# passwd {YOUR_USER_NAME}
# Here you just set the password like before, this time is for your user (not root)
```

Here's how it look like on my screen:

```
[root@archiso /]# useradd -m -G wheel -s /bin/bash Carthethyia
[root@archiso /]# passwd Carthethyia
New password:
Retype new password:
passwd: password updated successfully
[root@archiso /]#
```

Once you're sure you have added the user(s) you want, then you're good to go!

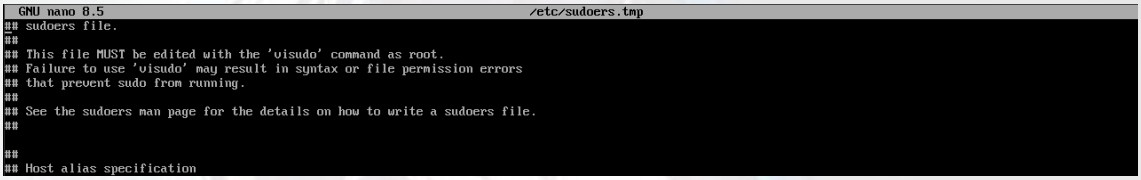
2.8. Sudo Setup

What exactly is sudo? Sudo is a command that can take your normal user, and give it admin/root privileges. Why don't we just switch user? Well, because frankly that will take a long time, and it's not really safe if you're doing everything as an admin/root because

one mistake (i.e. typo) will crash your system, this is Arch Linux, afterall :D. This is basically how you add sudo privilege into the user(s) you trusted.

```
# EDITOR=nano visudo
```

Make sure that you write it exactly like that (without the leading '# '). That will then open your GNU nano interface, it will look like this:

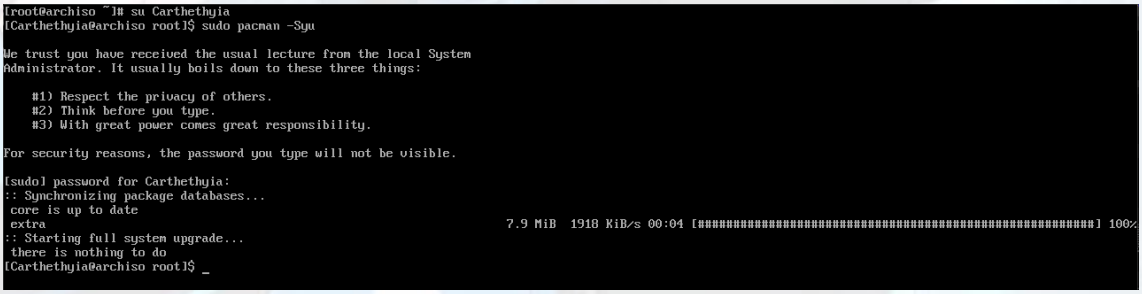


Like setting up the locale earlier, this time were looking for '**# %wheel ALL=(ALL) ALL**' and like before, we're going to uncomment it and make it like this '**%wheel ALL=(ALL) ALL**'. This will make it so that everyone that's on the 'wheel' group to run sudo commands. Make sure to write it before exiting the GNU nano interface.

To test if the user (not root) has sudo privilege you can do:

```
# su {YOUR_USER_NAME}
# sudo pacman -Syu <-- This will update the system
# Input your password here
```

If you ran this command and no error happened, then you're fine. If you're not sure, this is what happen when I run that command for the first time:



Once you're sure you have done what needs to be done, then you can exit from the user (go back to root) by doing:

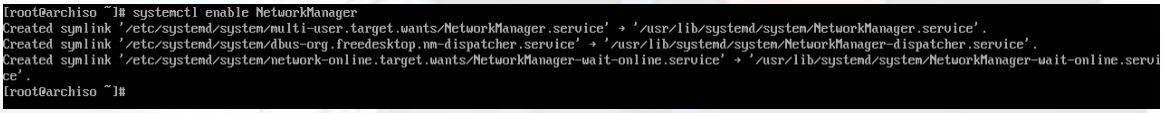
```
# exit
```

2.9. Services/Daemons

In this step, we're only enabling NetworkManager, so you can actually get network access once booting the system.

```
# systemctl enable NetworkManager
```

Make sure to write it exactly like that (without the leading '# '). This is what it looks like on my screen:



If your output has no error, then you should be fine for now, go to the next step!

2.10. GRUB

This is the last key step before rebooting. Setting up boot loader, GRUB.

```
# grub-install /dev/{DISK_PARTITION}
# list of examples:
# grub-install /dev/sda
# grub-install /dev/nvme0n1
```

But the usual UEFI command is:

```
# grub-install --target=x86_64-efi --efi-directory=/boot/efi \
  --bootloader-id=GRUB
```

I to be honest don't know which one is more correct, but you can refer to the official [arch linux GRUB guide] for further context.

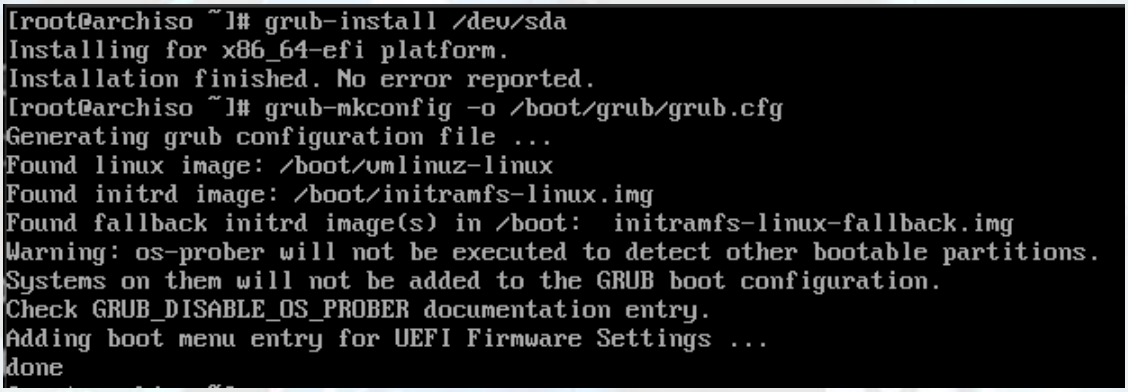
Next, we'll be configuring GRUB:

```
# grub-mkconfig -o /boot/grub/grub.cfg
```

MAKE SURE THAT THE '-o' IS NOT CAPITALIZE, BECAUSE I AIN'T DOING TROUBLESHOOTING FOR THAT.

You will face an error/warning that says something like "Warning: os-prober will not be executed to detect…" **and that is fine**.

Here's how it looks like on my screen:



2.11. Booting

We're now still in the system, so we need to exit into archiso by doing:

```
# exit
```

After that, your screen should have 'root' as a red text again. It should look like this:

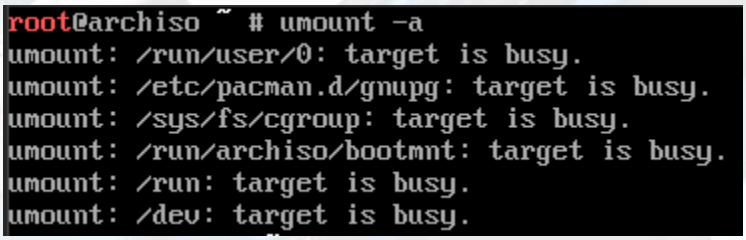
If somehow, you're not exiting into that, then make sure that you change your directory by doing:

```
# cd ~
```

Before then trying to exit again. We'll then unmount all of our drive that are not busy then rebooting, it should look like this:

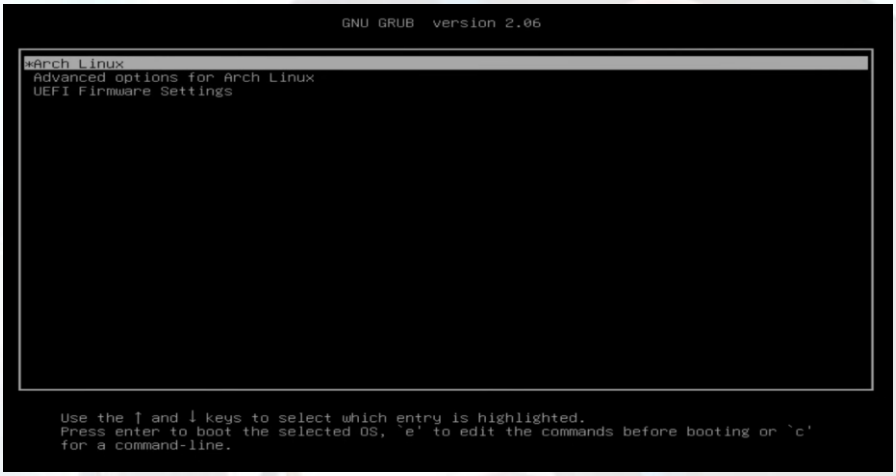
```
# umount -R /mnt
# reboot
```

Before rebooting, your screen will probably look like this:



And you know you're rebooting when your screen suddenly prints out many texts and turns off. This is where you pray to the Arch Linux god and hoping you're doing everything as intended.

After shitting your pants, you know you successfully boot into Arch Linux when your screen looks like this:



When it looks like that, congratulations! You've installed arch linux to your machine! After selecting Arch Linux, you will then be put into your login interface! There you should login as your user and not root with the password that you have set. Once logged in, you can go to the next chapter!

III. GUI and Packages

In this chapter we'll be installing and creating GUI, as well as installing packages (apps/application) that we need. We'll also be making sure if our system runs properly!

3.1. NetworkManager

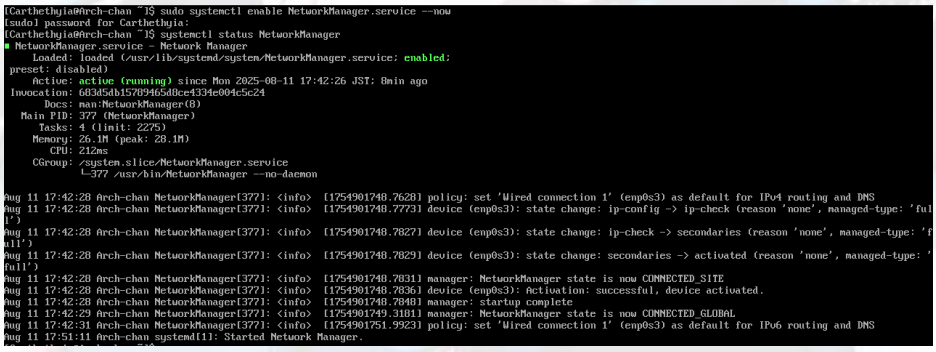
Now, we need to set up the internet AGAIN… If you're plugged into ethernet, you should be fine, but we're going to check anyways:

```
# ping -c3 google.com
```

If that fails, it's fine :v. We're going to set up NetworkManager to work with wifi again.

```
# sudo systemctl enable NetworkManager.service --now
# systemctl status NetworkManager
```

Your screen should then look like:



If it looks like that, you're good! Next we'll do almost the same stuff like the ones before.

```
# nmcli device wifi list <-- List available network
# nmcli device status <-- Look for wlan or wlp2s
```

Connecting to your wifi using:

```
# nmcli device wifi connect {SSID_NAME} password {PASSWORD}
# nmcli connection show --active <-- to verify
```

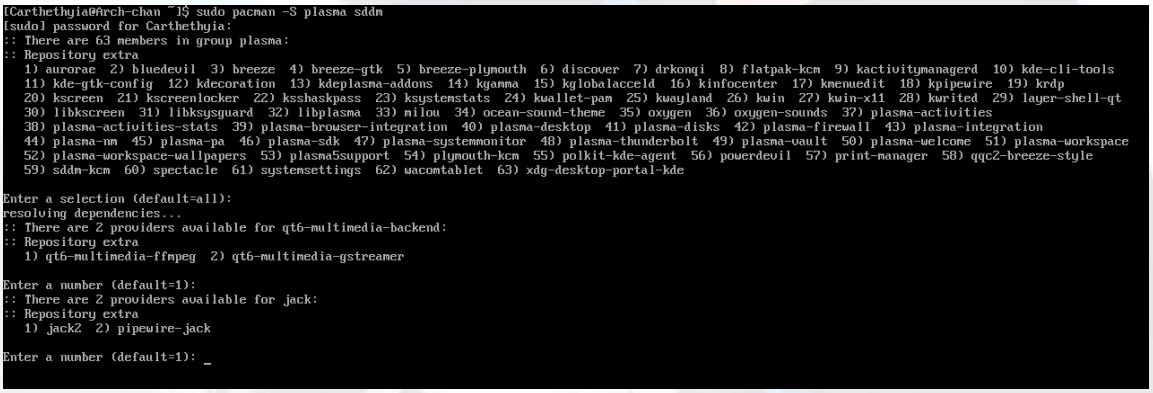
After all of that, test your connection again by pinging some public domain like the example above.

3.2. Installing Packages pt.1

In this step, we'll be setting up the basic GUI environments. A pretty good graphical environment that we'll be installing is KDE plasma. This also comes in a packet with wayland, which will come handy further on if you're planning to do some ricing.

```
# sudo pacman -S plasma sddm konsole firefox
```

Here’s how my screen looks:

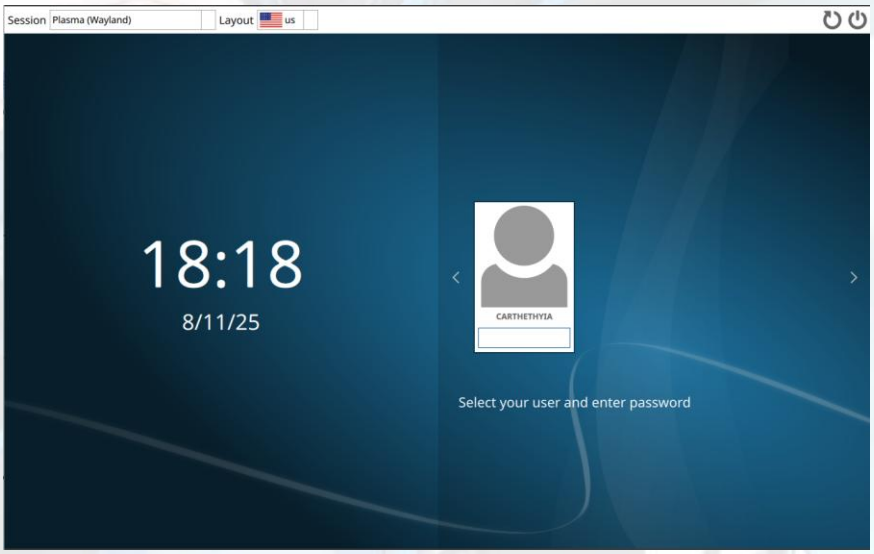


After doing the commands and inputting your password, you will just let it run with the default settings by always pressing ‘Enter/Return’. At the end, proceed with installation and wait again. You can now take a shower while we’re waiting for it to be downloaded 😊

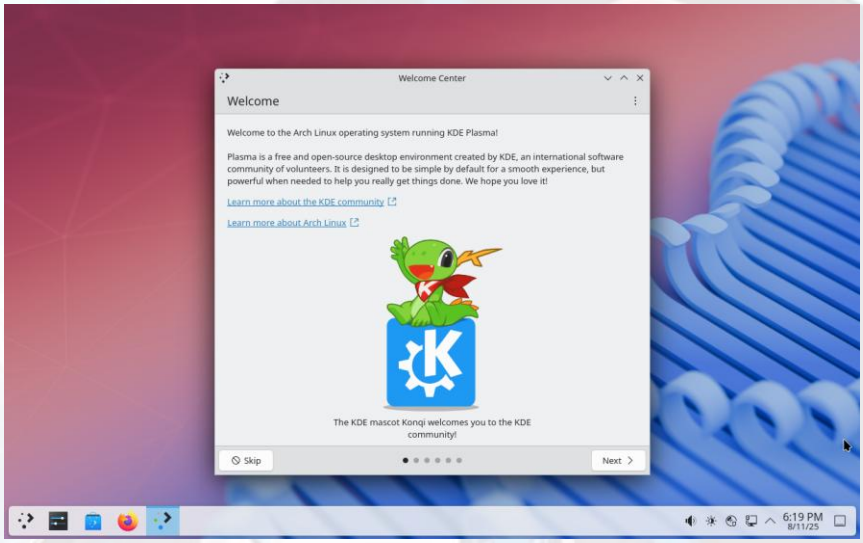
3.3. Launching Display

```
# sudo systemctl enable --now sddm
```

After running that, your system should look like this:



After logging in, it should look something like this:



You can just press skip and you'll be fine.

3.4. Making Sure Everything Works pt.1

3.4.1. Make sure internet is working

This could be done by opening youtube.com from firefox (or any browser you chose) and seeing if you can connect to https://www.youtube.com/@Rhio_Bimo. While you're there, you can also try signing in and testing if you can fully utilize the google feature by also subscribing to the youtube channel.

3.4.2. Make sure that sounds is working

After that, you would also probably see a video about installing arch linux, why don't you go there and watch the video to see if the sound is properly working. If it does, might as well drop a like towards the video!

3.4.3. Make sure console is working

There's a default keybind for opening a console, that is 'ctrl+alt+T' if it pops up konsole, then thank god, ur good and you can go to the next step!

3.5. Installing Packages pt.2

The packages that we'll be installing here are: vscode, dolphin, yay (AUR), git, wine, mousepad, nomacs, mpv, oracle vbox, and docker.

To do this, we need to configure our pacman. To do that you can open your konsole and do:

```
# sudo nano /etc/pacman.conf
```

You're probably a pro now, so in the GNU nano, now we'll looking for something like this:

```
[core]
Include = /etc/pacman.d/mirrorlist

#[extra-testing]
#Include = /etc/pacman.d/mirrorlist

[extra]
Include = /etc/pacman.d/mirrorlist

# If you want to run 32 bit applications on your x86_64 system,
# enable the multilib repositories as required here.

#[multilib-testing]
#Include = /etc/pacman.d/mirrorlist

#[multilib]
#Include = /etc/pacman.d/mirrorlist
```

We'll be un-commenting both the '#[multilib]' and '#include = /etc/pacman.d/mirrorlist'. Once done, just write it and you should be done for the pacman configuration.

You will then refresh the pacman package by doing:

```
# sudo pacman -Sy
```

We'll now install the build tools for AUR:

```
# sudo pacman -S --needed git
```

After that we'll be installing yay:

```
# cd ~
# git clone https://aur.archlinux.org/yay.git
# cd yay
# makepkg -si
```

Test our yay:

```
# cd ~
# yay --version
```

If it says the version of yay, then Yay! You're goof! Next!

Now, we'll finally install the rest of the packages! Be sure that you have enough storage for all these packages (estimated 16GB):

```
# sudo pacman -S --needed code dolphin wine winetricks\
  mousepad mpv virtualbox docker ffmpeg \
  gst-libav gst-plugins-good gst-plugins-bad \
  gst-plugins-ugly
```

Some of the package that's not been installed are nomacs, for it, we'll now use yay:

```
# yay -S nomacs
```

For yay, just stick to the default and select '[A]ll' if prompted. You may have to keep an eye on your installation while in yay because when it prompts you to put in a password it could time out (you can technically set this up so it doesn't do that, but I'll now be showing you it in this document).

Now I'll be listing what each of those packages do:

- yay : For installing packages that are not available through pacman
- git : To pull and push into git (this is essential if you're using linux)
- code : Visual Studio Code (you can also use other like kate or even vim)
- dolphin : Basically, windows file manager, so you don't have to always use CLI to look through your files
- wine : To install application that are not native to Linux (i.e. LINE)
- mousepad : A simple text editor, like Notepad in Windows
- mpv : Video player and encoder
- virtualbox : If you want to create a virtualization environment for other OSes
- docker : If you want to create a containerization for development
- ffmpeg : for mpv
- nomacs : To display and edit image

3.6. Making Sure Everything Works pt.2

For each of the packages we've installed, we're going to test if they're functioning correctly:

- yay : If you can install nomacs through yay, you're good
- git : If you can install yay, you're good
- vscode : In your konsole, type 'code' and enter. It will open code OSS. Go to the extension tab, if it loads, you're good
- dolphin : Default keybind is 'WindowsKey/Master+E', if it opens and you can see your files, you're good
- wine : In your konsole, write 'winecfg' and if it prompts you to download wine mono and actually create a '.wine' folder in your Home directory, you're good.
- mousepad : Type 'mousepad' in your konsole, if it opens the program, you're good
- mpv : Download a video somewhere on the internet, and try to play it in mpv, if you can play it, you're good (try it also with mp3/audio files)
- virtualbox : Type 'virtualbox' in your konsole, if it runs and when you open the window the notification compiles, you're good
- docker : Type 'docker --version' and if it returns a version, you're good!

- Nomacs : Download this image. If you can open it with nomacs, then you're good!
- The other : It's for mpv, so if you mpv runs, it's probably fine :v

3.7. Cleaning Excess Files/Packages

After installing many packages, your files will be clutered with a bunch of unnecessary/ temp files. So now we're going to clean it up a bit!

```
# yay -Sc --aur <-- remove cached AUR package
# sudo pacman -Sc <-- remove pacman cache
# sudo pacman -S pacman-contrib <-- Install pacman cleaning tools
# sudo paccache -r <-- keep only 3 recent versions
# sudo paccache -rk1 <-- keep only 1 recent version
# sudo paccache -rku0 <-- keep only recent versions (use at your
own risk)
# sudo pacman -Rns $(pacman -Qtdq) <-- cleaning unused-
dependencies
# sudo journalctl --vacuum-time=3weeks <-- to clear any unused
files if it reached 3weeks
```

With those commands, you should have a clean linux setup with most of the basic packages installed. You can also go to your '~/.cache' directory and delete stuff from over there.

3.8. You Use Arch, BTW

It wouldn't be an Arch Installation guide if it were only to stop here. For our final 'necessary' guide, we'll be installing fastfetch (or neofetch).

fastfetch :

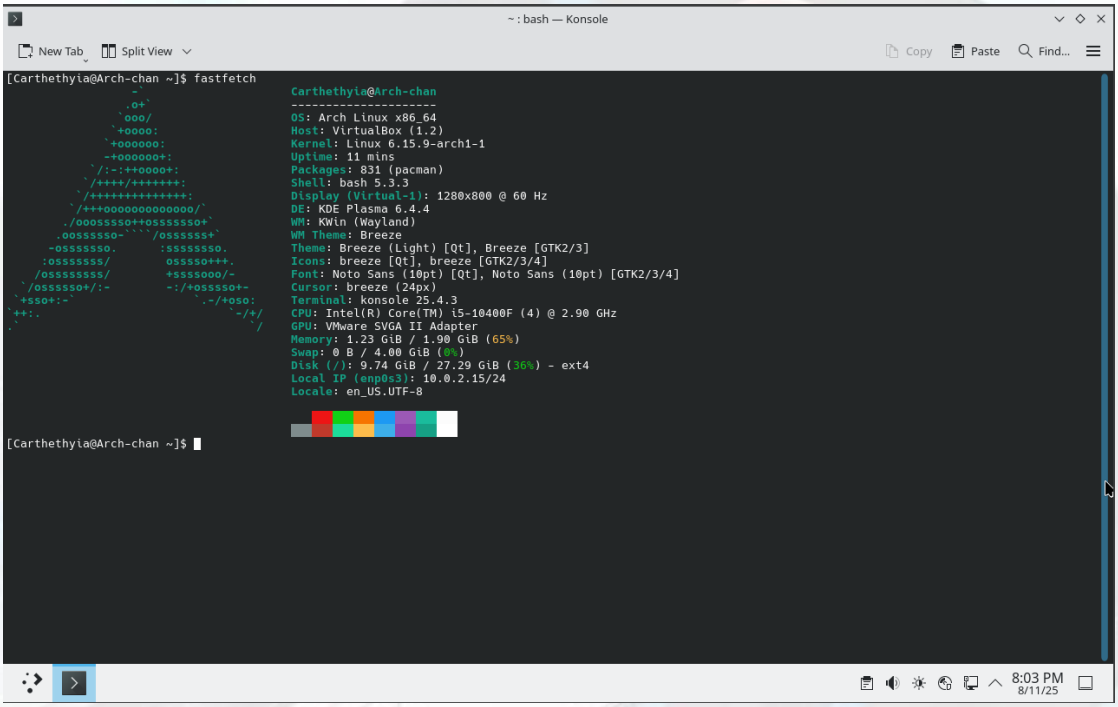
```
# sudo pacman -S fastfetch
```

neofetch:

```
# yay -S neofetch
```

Run it:

```
# fastfetch {or 'neofetch'}
```



With this screen, you're basically done installing Arch Linux on your machine and can stop reading here! If you want to know how the packages installed is used or other Quality of Life (QOL) functionality, then you can continue reading!

This has been Rhio Bimo (Cola1000) and thank you~!

IV. Ricing and QOL

From this point on, everything will be optional and 'just for fun.' If this is your first time also installing any kind of Linux distros, then congratulations on making it to here! I'm also assuming you're already pretty familiar with the Arch Linux's environment (at least as much as me {which is not a lot, tbh :v})

4.1. Setting Up DNS over HTTP(S) / DoH & Firewall

DNS over HTTPS (DoH) is a protocol that sends DNS queries over an encrypted HTTPS connection instead of plain text. This prevents eavesdropping and manipulation of DNS requests, improving privacy and integrity. It does not replace a VPN or encrypt general web traffic; it only protects the domain name resolution step.

First test:

1. Split your terminal and do these:
    a. Terminal 1: Run a tcpdump command on port 53

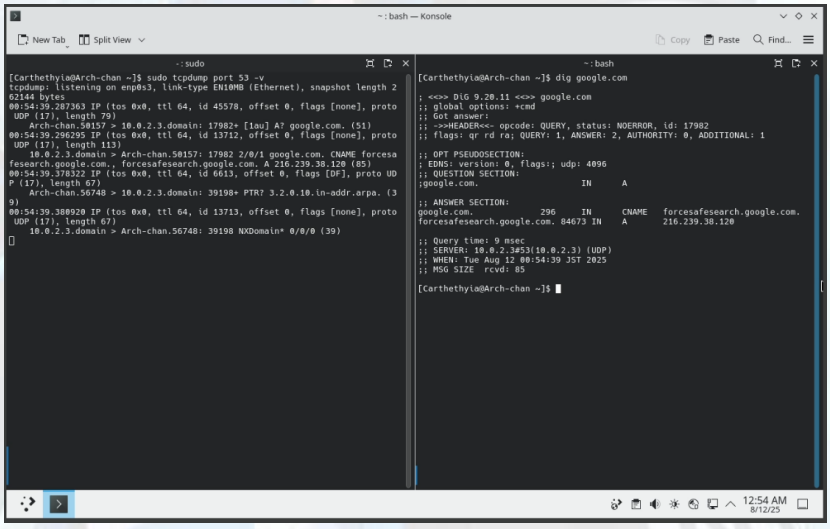
```
# sudo tcpdump port 53 -v
```

    b. Terminal 2: Run a dig command on any website

```
# dig google.com
```

    Quick note, the package needed to run these commands are: tcpdump and bind

Your screen will probably look like this:



From the image, we can see that whenever we dig for something, another user can see through what we're digging. This is obviously a concern of privacy.

Now, we're going to set up DoH:

1. Open systemd's resolve.conf

```
# sudo tcpdump port 53 -v
```

2. Edit it:
    a. Find the cloudflare thingy, and basically **cut**: "1.1.1.1#cloudflare-dns.com" (from the right side of the '=' symbol) **from** "# Cloudflare=1.1.1.1…" and paste it in line below which says "#DNS="
    b. Uncomment that line too
    c. Uncomment the line that says 'DNSOverTLS=no' and change the 'no' to 'yes'
    d. Uncomment the line that says 'MulticastDNS=yes' and change it to 'no'
    e. Write and Quit.
3. Make a back up and delete:
    a. Make a back up of the resolv.conf

```
# sudo cp /etc/resolv.conf /etc/resolv.conf.old
```

    b. Remove the resolv.conf

```
# sudo rm /etc/resolv.conf
```

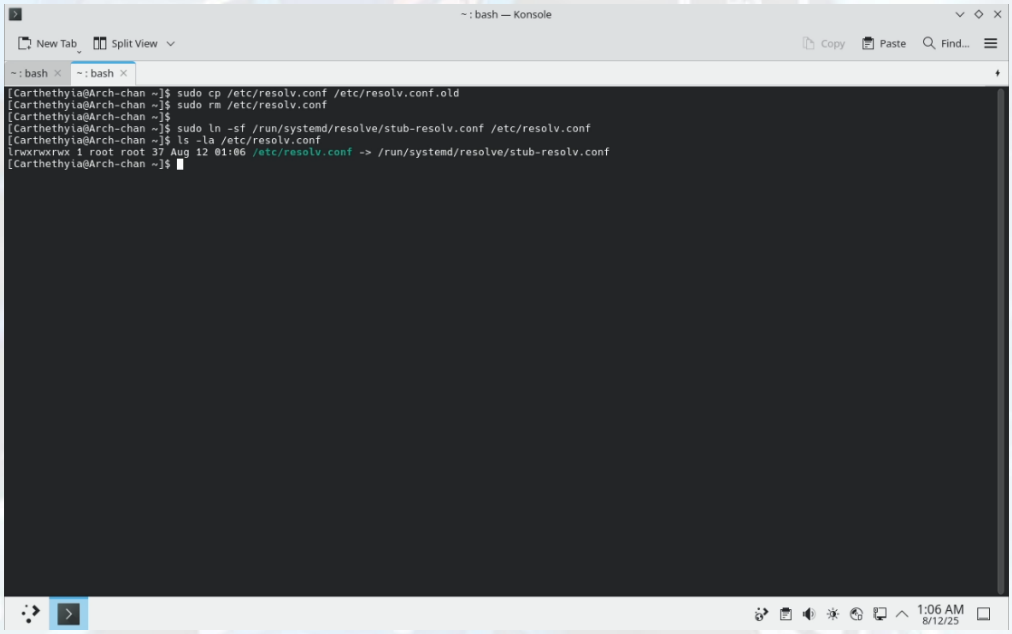
4. Make symlink:
    a. Make the symlink

```
# sudo ln -sf /run/systemd/resolve/stub-resolv.conf \
  /etc/resolv.conf
```

    b. Test the symlink

```
# ls -la /etc/resolv.conf
```

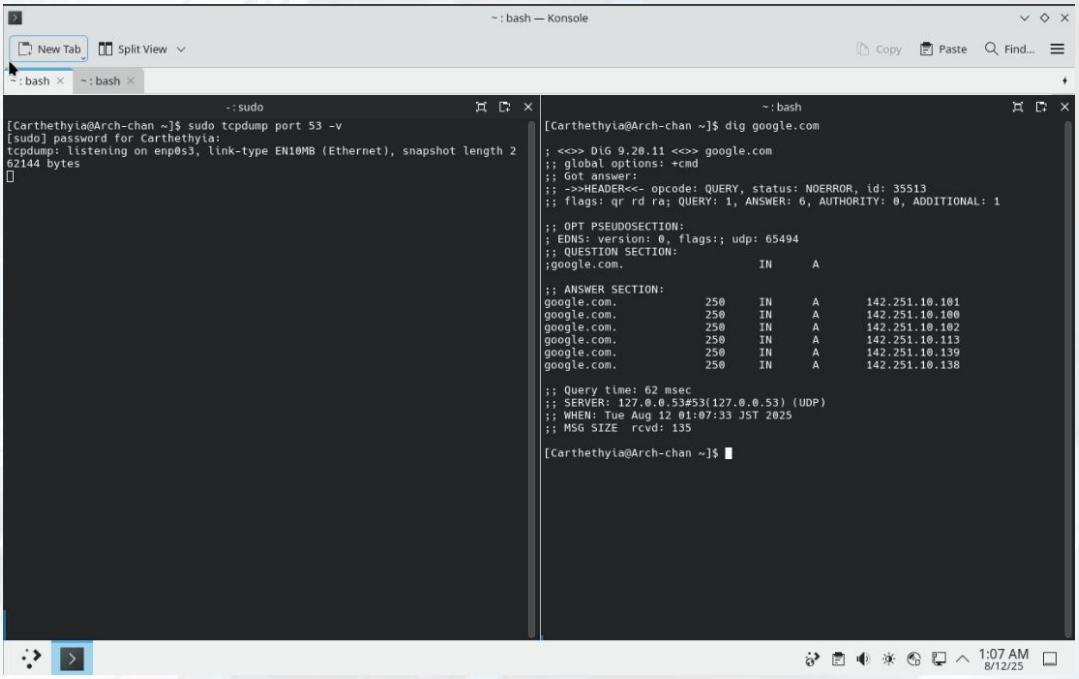
    Your screen will probably look like this:



5. Enable it:

```
# sudo systemctl enable systemd-resolved --now
```

6. Test it again:
   Just do it like the previous one (two terminal, one tcp dump, other dig) and if you did it correctly, it will look like this:



   Which you can see that when we're digging, it doesn't get scanned by the tcpdump!

A firewall is a network security system that monitors and controls incoming (inbound) and outgoing (outbound) network traffic based on predefined rules. It can block specific applications, IP addresses, ports, or protocols to protect the system from unauthorized access and data leaks. This is very useful if you're running a 'dangerous' program that can sends out private information via internet (~~adobe~~).

1. Installing firewall

```
# sudo pacman -S ufw
# sudo systemctl enable --now ufw
```

   The full documentation for ufw will be provided **here.**
2. Blocking and Enabling Rules
   a. Default

```
Here are some examples:
# sudo ufw default deny incoming
# sudo ufw default allow outgoing
```

   This means that the default firewall will deny any incoming connection but allow itself to send out something.

b. Blocking

```
Here are some examples:
# sudo ufw deny 80/tcp <--Deny port 80 (HTTP)
# sudo ufw deny 443/tcp <--Deny port 443 (HTTPS)
```

c. Allowing

```
Here are some examples:
# sudo ufw allow ssh <--Allow basic service in ssh
# sudo ufw allow 443/tcp <--Allow port 443 (HTTPS)
```

3. Enable Firewall

   You can enable firewall and see the status as well as the firewall rules you've set by doing these commands.

```
# sudo ufw enable
# sudo ufw status verbose
# sudo ufw status numbered
```

4. Testing

   Depending on your rule, you can test it via curl and ping.

5. Reseting Firewall

```
# sudo ufw reset <--This will reset all rules
```

6. Firewall w/ GUI

   If you're using plasma KDE (I think it should also work with other GUI packages, I only install this one, so I don't know), if you go to the system settings, you'll also see firewall settings there. In there you can intuitively set up any rules you want just like using ufw (but I think you still need to install the ufw package, idk).

4.2. Downloading Programing Languages

Some user might want to install many different programming languages that they want to use, so here's some way to install a few of them. Remember, being an Arch user usually means you got to read a lot of documentation, and basically all programming language will have Linux support, so you can (and should) read the installation guide from the official documentation provided by the language.

```
# sudo pacman -Syu
```

This is for updating the package database.

4.2.1. gcc/g++

```
# sudo pacman -S gcc
# gcc --version
# g++ --version
```

```
//test code gcc:
#include <stdio.h>
int main() {
```

```
//test code g++:
#include <iostream>
using namespace std;
```

```
    printf("Hello from C!\n");
    return 0;
}
```

```
int main() {
    cout << "Hello from C++!" << endl;
    return 0;
}
```

```
Compile:
gcc test.c -o test_c
./test_c
```

```
Compile:
g++ test.cpp -o test_cpp
./test_cpp
```

4.2.2. python

```
# sudo pacman -S python python-pip
# python --version
```

```
// Test Code + Compile:
python -c "print('Hello from Python!')"
```

4.2.3. JDK

```
# sudo pacman -S jdk-openjdk maven gradle
# javac -version
# java -version
# mvn -v
# gradle -v
```

```
// Test Code:
public class Test {
    public static void main(String[] args) {
        System.out.println("Hello from Java!");
    }
}
```

```
Compile:
javac Test.java
java Test
```

4.2.4. Go

```
# sudo pacman -S go
# go version
```

```
// Test Code:
package main
import "fmt"
func main() {
    fmt.Println("Hello from Go!")
}
```

```
Compile:
go run test.go
```

4.3. VM and Container

Virtual Machines (VMs) emulate an entire hardware system, allowing you to run a full operating system inside another. Tools like VirtualBox let you install and run OSes in isolation from your main system. Though linux is the superior operating system, you don't want to necessarily install all of them, here you will learn how to set up a virtual machine in Oracle Virtual Box.

1. Download the ISO for the OS you want
2. Open Oracle Virtual Box and click 'New'
3. Setup:
    a. Name : Name of the VM
    b. Folder : The folder which the VM sits
    c. ISO Image : Put your downloaded ISO image there
    d. Editiion : Idk, left it empty
    e. Type : Your OS type (Linux, MWindows, Solaris, etc.)
    f. Subtype : Subtype of your OS (i.e. Arch Linux)
    g. Version : Your OS's Version (i.e. Arch Linux 64-bit, Windows 11, etc.)
4. Click next and chose the Base Memory (I suggest at least 1GB) and CPU processor (At least 1 core)
5. Enable EFI is for OS that supports EFI
6. Hard Disk:
    a. Create Virtual Hardisk, select the folder which it sits
    b. Give it some memory, this will be your VM's storage
7. Settings:
    a. System : If you enable EFI, it's fixed. If you didn't make sure 'Optical' is above 'Hard Disk'
    b. Storage : Select the 'Empty' from 'Controller IDE', click the disk icon on the right, select your ISO image, check the 'Live CD/DVD'
    c. Network :
        i. NAT : Safely give VM internet without exposing it to LAN
        ii. Bridge : When VM needs to act like a real machine on the LAN
        iii. IntNet : Isolated env, malware test, or multi-VM simulations.
        iv. Etc : FAFO
    d. Etc : FAFO
8. Click 'Ok' and the 'Start'

Other than virtualization, there's also containerization. Containers provide a lightweight way to package and run applications in isolated environments, sharing the host OS kernel

but maintaining separate user spaces. This makes them faster to start and more resource-efficient than VMs.

1. Enable Docker Service

```
# sudo systemctl enable --now docker
```

2. Allow User to Run Docker Without sudo

```
# sudo usermod -aG docker $USER
```

3. Test Docker

```
# docker run hello-world
Output: "Hello from Docker!"
```

4. Docker Compose

```
# sudo pacman -S docker-compose
# docker-compose --version
```

5. Basic Docker Commands

Command	Description
`docker ps`	List running containers
`docker images`	Show downloaded images
`docker run -it ubuntu bash`	Run an Ubuntu container interactively
`docker stop <container_id>`	Stop a running container
`docker rm <container_id>`	Remove a container
`docker rmi <image_id>`	Remove an image

There are many differences between VM and container, but just to be quick, container is a much more lightweight version of virtualization. That doesn't mean that VM is bad, VMs generally offer stronger isolation and can run entirely different OS types, making them more versatile for certain security and compatibility needs. Containers are ideal for software development, testing, and deploying applications consistently across different systems.

4.4. Wine Is Not an Emulator (wine)

Wine is a compatibility layer that lets you run Windows applications on Linux (and other Unix-like OSes) without needing a Windows VM. Here you will learn how to install a program that's not linux compatible (LINE).

Note: From my experience, you kinda need an older version of LINE to run, you can then run the later version (?) Tbh, idk how to replicate it, but this is the steps I've take:

1. Make sure you have wine and winetricks

```
# sudo pacman -S wine winetricks
```

2. Download LINE from web through this link

3. Prepare Wine Prefix

```
# WINEPREFIX=~/line-wine wineboot
```

4. Install Required Libraries

```
# WINEPREFIX=~/line-wine winetricks ie8 corefonts wininet
```

5. Install LINE into wine

```
# WINEPREFIX=~/line-wine winecfg
# Select the windows 10
# WINEPREFIX=~/line-wine wine {YOUR_LINE.exe_PATH} /skip-proxy
# Maybe also try this:
# WINEPREFIX=~/line-wine-old wine reg add \
  "HKCU\Software\Wine\Mac Driver" /v DisableIPv6 \
  /t REG_SZ /d y /f
```

6. Launching LINE

For the first time, LINE will probably start automatically. But for the following launch:

```
# WINEPREFIX=~/line-wine wine \
  "C:/Program Files (x86)/LINE/LINE.exe"
```

If you want to make a desktop shortcut:

```
# nano ~/.local/share/applications/line.desktop
```

```
[Desktop Entry]
Name=LINE (Wine)
Exec=env WINEPREFIX=/home/{YOURUSERNAME}/line-wine-old wine
"C:/Program Files (x86)/LINE/LINE.exe"
Type=Application
StartupNotify=true
Icon=wine
Categories=Network;Chat;
```

```
# chmod +x ~/.local/share/applications/line.desktop
```

4.5. Customizing Bootloader

When you're using linux and starting up your machine, you would probably see a lot of text scrolling. Here, I want to get rid of that and instead put a cool animation when my system boots. We'll be using Plymouth (or View in Github here).

```
# sudo pacman -S plymouth plymouth-kcm
# yay -S plymouth-theme-NAME-git <--View the NAME in GitHub
```

To apply it, go to your KDE system settings and search for 'Global Theme', go inside and find 'Boot Splash Screen'. Your theme should be in there.

You'll also need to edit the grub loader file.

```
# sudo nano /etc/default/grub
```

You'd then find "GRUB_CMDLINE_DEFAULT=…" and make sure that the right side of the equal sign has 'splash' in it (i.e. GRUB_CMDLINE_LINUX_DEFAULT="quiet loglevel=3 splash") if there isn't, add it.

Make sure you also update grub:

```
# sudo grub-mkconfig -o /boot/grub/grub.cfg
```

You can then try rebooting your system.

4.6. Secure Boot

As the name suggest, secure boot is a UEFI firmware feature that ensures only trusted, signed bootloaders and kernels can be executed when your system starts. Its main goal is to protect against bootkits or malware that loads before your OS.

Note: You will most likely brick your OS while learning how to do secure boot, so please proceed with your own risk.

<I don't have enough time to do this, sorry>

4.7. Ricing

In the Linux community, ricing means customizing your desktop environment or window manager's look and feel, much like "pimping" your ride, but for your Linux UI. The term likely comes from "rice rockets" (slang for highly customized Japanese tuner cars {and ofc it's Japanese :v}), and in Linux it just means making your setup look unique.

There are many tools if you want to do some ricing, but because KDE plasma already came with wayland, I'm going to use hyprland to rice-up my setup. In this tutorial, I'm only going to show you how to change your background and have 'cool' waybar.

Install the packages:

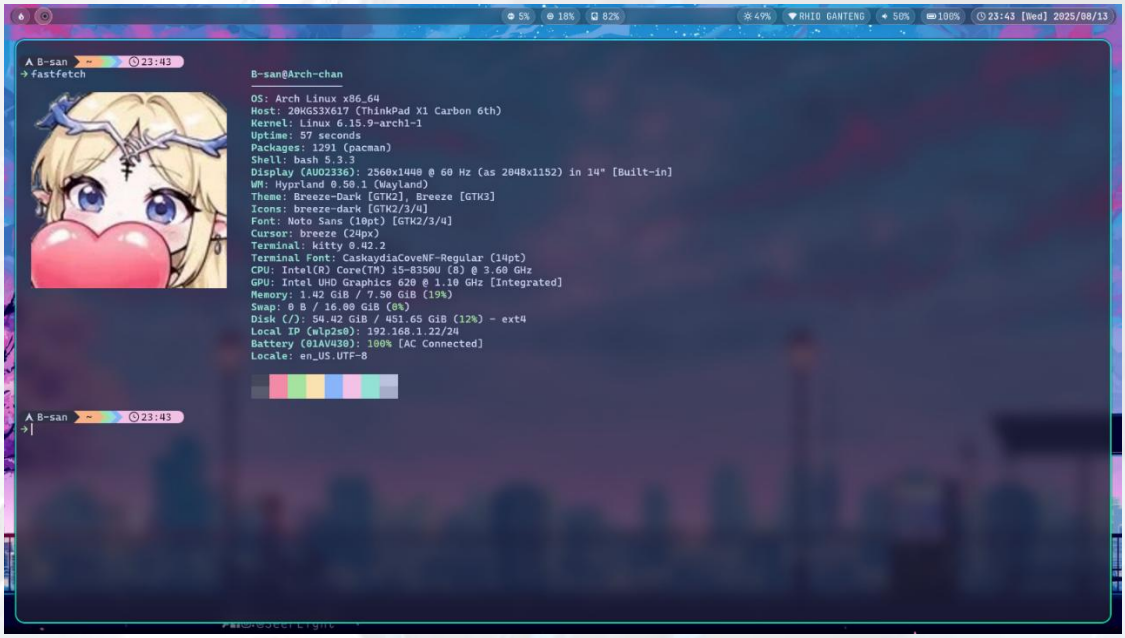
```
# sudo pacman -S hyprland kitty wofi waybar ttf-font-awesome \
  ttf-cascadia-code-nerd
# yay -S hyprshot swaync hyprlock hypridle stow hyprpaper starship
```

The next step is pretty simple but can be quite confusing. I ain't walking you through my whole ricing journey, so you can just take the configuration from my github (here). But if you're skeptical, you can watch this playlist by typecraft that uses hyprland to rice.

If you're using my github repo, make sure to unzip everything from RIcingConf (yes, you should just download the zip from there) into '~/.config/' and you should be fine (I think). Make sure you read the key binding first so that you're not too confused.

You should log out (if you follow this guide all the way through, then it’s from your KDE plasma). Before you log back in, you can see somewhere on the screen that you can change the GUI. You should change into plasma.

The ricing screen should look like this:



If your screen doesn’t look like this, idk, maybe look into the config and make sure the monitor/screen name actually matches with yours (or something, idk) ask AI or something.

V. Extras

At this point, you're just here for fun, to check if I made a mistake (which I did, a lot too), and/or if you want to see what kind of things you can do with the newfound power. I'll just be brief in this part of the document because you already know how Arch works.

5.1. Watching Anime in Command Line

Yes, this is how Arch user watch anime:

1. Clone the repository

```
# git clone https://github.com/pystardust/ani-cli.git
```

2. Copy it to local/bin

```
# sudo cp ani-cli/ani-cli /usr/local/bin
```

3. Launch it

```
# ani-cli
```

Note: need fzf

5.2. Creating your own Distros/ISO

Yes, you can create your own custom image file and even custom distros for other people to use! Make sure you have enough memory if you want to wrap everything we've build here, because I sure don't have it :v.

Download the archiso package, you can also see the documentation here.

```
# sudo pacman -S archiso
```

I ain't got enough time to do this so I'm sorry, but if you can read my document until the end, then maybe read the actual document from Arch Linux org.