Debian Linux `netinst` RAID Install

with terrible pictures

By Lawrence Candilas – https://github.com/lawrencecandilas

Contents

Introduction

What follows is a walkthrough on installing Debian from the `netinst` .ISO on to a pair of 128GB SSDs in a RAID-1 configuration.

This is a fairly involved and detailed guide and will require a decent level of concentration. There may be easier ways to accomplish this but this is my method I've come to rely on. I welcome any feedback.

What's so special about a 128GB RAID-1?

Are you building a Linux PC that's meant to be appliance-like, and really just need a place for the operating system to live, and maybe a single app or two?

A 128GB RAID-1 is a good strategy because:

- Appliance/embedded-like use cases don't need a lot of storage for the Linux OS.
  8GB is really enough.
    - 128GB SATA SSDs seem to be the cheapest on Amazon right now as of February 2023.
    - SATA SSDs are better than things like USB flash drives or SD cards:
        - SATA SSDs perform better due to reliable TRIM support.
            - I don't know how to tell if this is a thing on modern SD cards or USB flash drives.
        - SATA SSDs cannot be accidentally removed unless someone opens the chassis.
        - The SATA interface is designed to be non-removeable.
            - Anecdata: I have had weirdness on some older Dell systems where USB devices randomly disconnect and reconnect. It could have been a hardware issue.
- Appliance/embedded-like use cases need reliability:
    - OS and basic app storage should be protected against single device failure.
        - Hence, the RAID-1 – and I mean Linux software RAID-1, not motherboard RAID.
        - Linux supports and boots just fine off of a RAID-1.
        - Putting GRUB on all drives and configuring firmware properly will allow the system to boot and run if one drive dies.
    - SSDs are not sensitive to motion.

What's so special about Debian `netinst`?

The Debian `netinst` is as barebones as you can possibly get *while* still having access to Debian's famous and well-supported repositories; which are widely used and respected.

Right after the base install, nothing else will be installed other than the bare minimum to get a useable system. The next step is to select what you need from literally tens of thousands of Debian packages to choose from and install from the mirros. External `.deb`'s can of course be installed as needed.

Now, the intent of `netinst` is to allow installation over the Internet without having to download all of the CDs or DVDs that make the entire distribution. You are able to select "tasks" or particular collections of packages, then it downloads and installs straight from the Internet.

Of course you do not have to select any “tasks”: for a true, minimal, barebones system – you simply select nothing.

There are more minimal ways to do Linux (e.g. Linux from Scratch, OpenWRT) but they aren’t as easy to keep updated nor do they have easy access to the variety of packages that are in the Debian repos.

You will have to hunt for device firmware though – such as blobs required by certain Wi-Fi or hardwired Ethernet NICs. ☹

Let's go!

LEGACY MODE for UEFI systems

The below steps were done on a UEFI-based system in Legacy mode. To do that, at least for the system I was using to write this document:

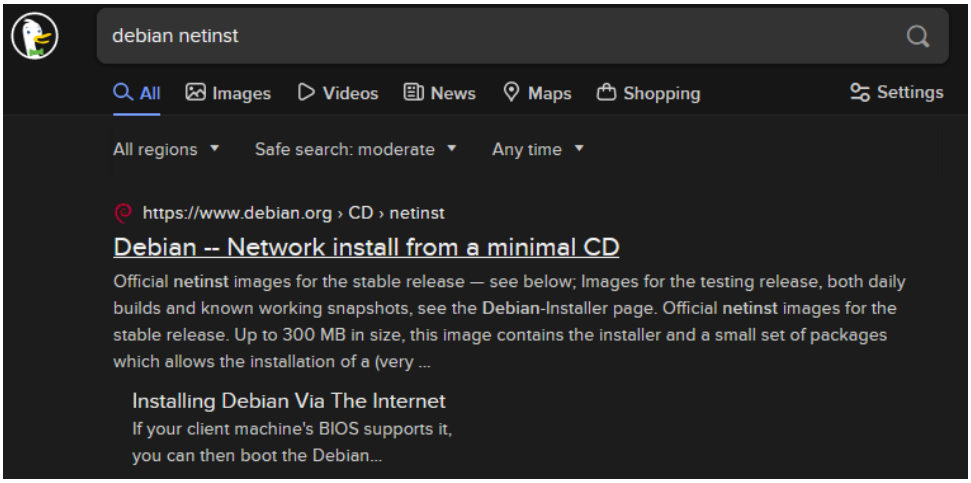
- Disable all UEFI boot options
- Make sure the boot option that the USB flash drive will hit will be a legacy mode option.

UEFI would be the better choice if I could get it working. Windows 11 has dropped support non-UEFI systems (because TPM 2.0 won't work in "CSM" BIOS compatibility mode), so it will only be a matter of time before PCs no longer support Legacy mode.

Details about me getting this working with UEFI will be in a future document.

Getting The Bootable `netinst` image.

Entering "Debian netinst" in your favorite search engine should take you to the right place, currently `https://www.debian.org/CD/netinst/`.



and the amd64 CD image is what we want – and we'll download it directly (it's not that big).

Despite the fact it's called a CD Image, it can be applied to a USB storage device to make a bootable USB device.

Creating Bootable Media

On Linux, my favorite tool for that is the `dd` command.

On Windows, I typically use Rufus (https://rufus.ie/en/).

Configure your PC to boot from USB or removeable media, and disable Secure Boot.

This will be specific to your PC's firmware.

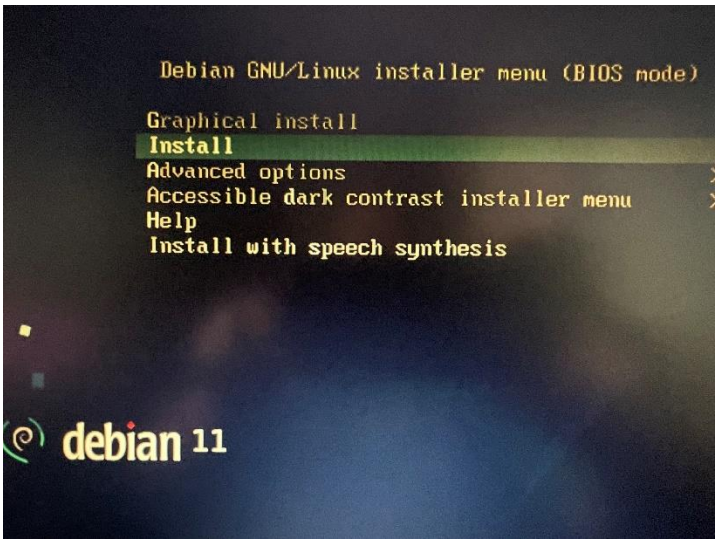
Secure Boot will just get in the way. Disable it. A future document may cover doing this with Secure Boot in the future.

Boot from the USB flash drive with the installer on it

You will see the lovely Debian installation screen.

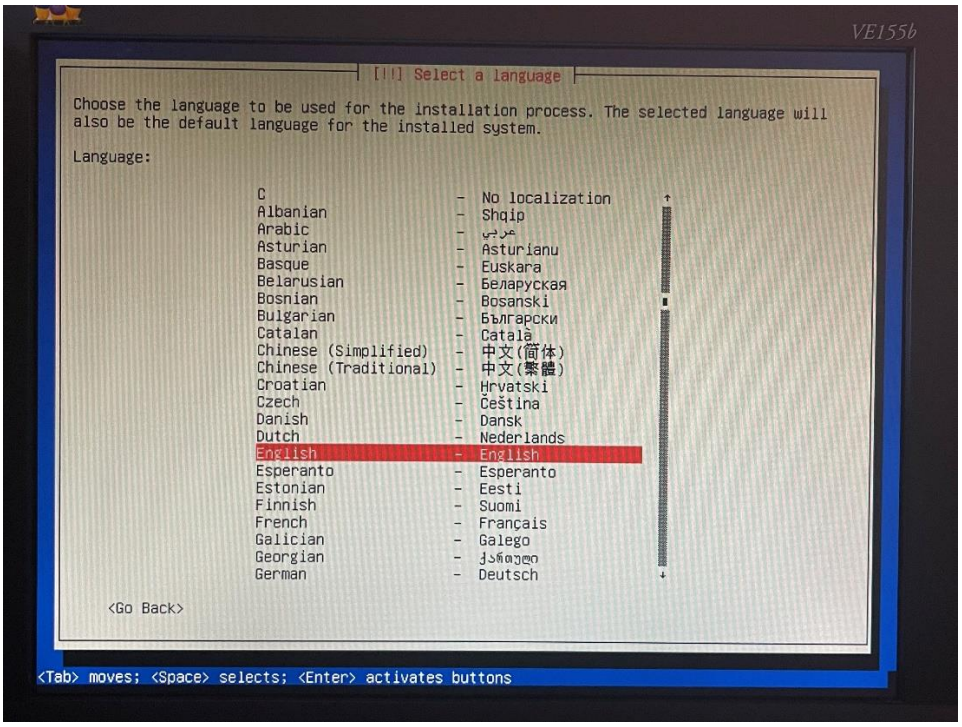
If it doesn't say BIOS mode, this guide won't work.

Move down to "Install" and hit Enter. We won't use the "Graphical install."

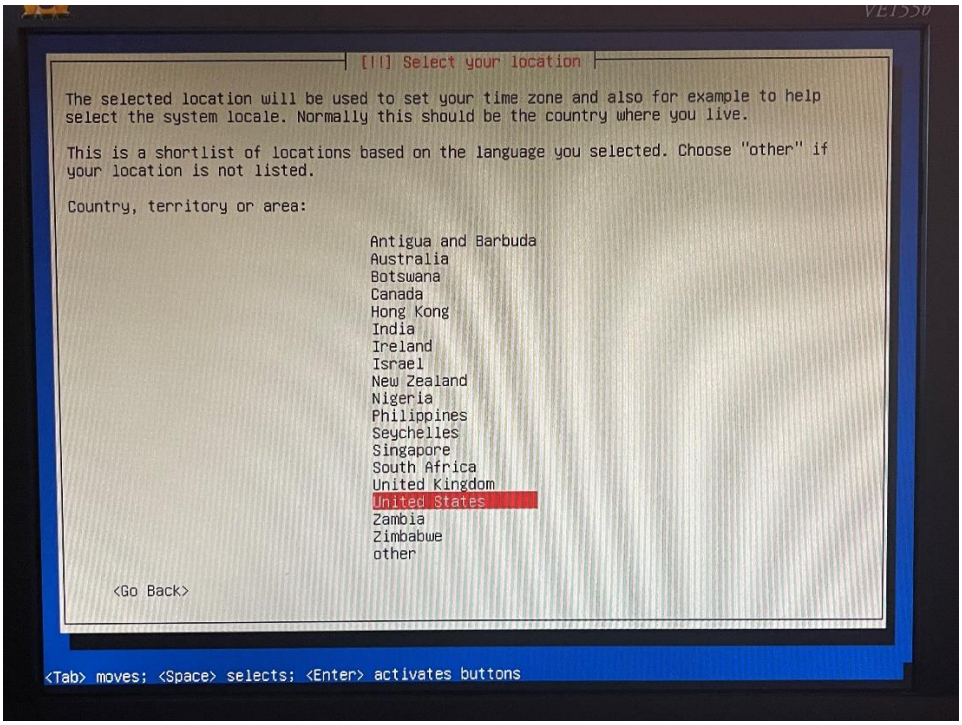

Debian GNU/Linux installer menu (BIOS mode)
Graphical install
Install
Advanced options
Accessible dark contrast installer menu
Help
Install with speech synthesis
debian 11

Initial installer questions

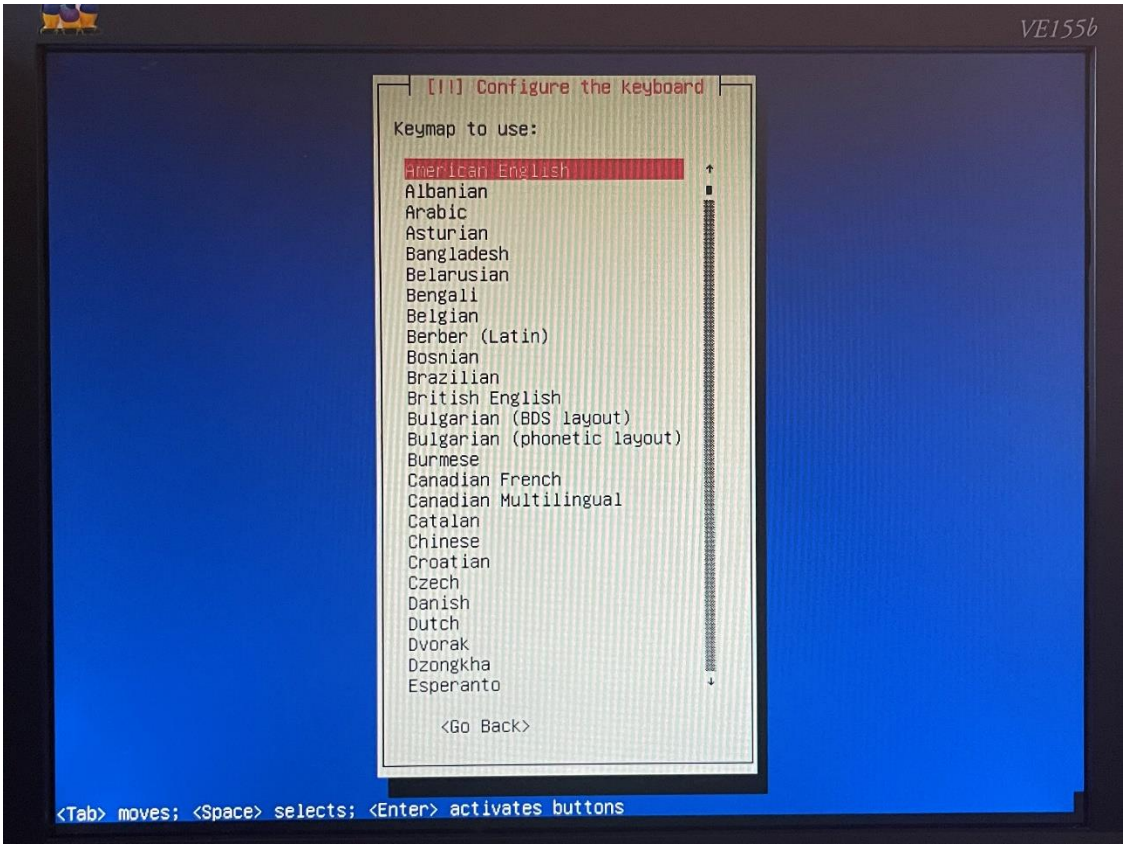
I select “English” here – select any language you understand. This guide is in English so you’ll see English here.



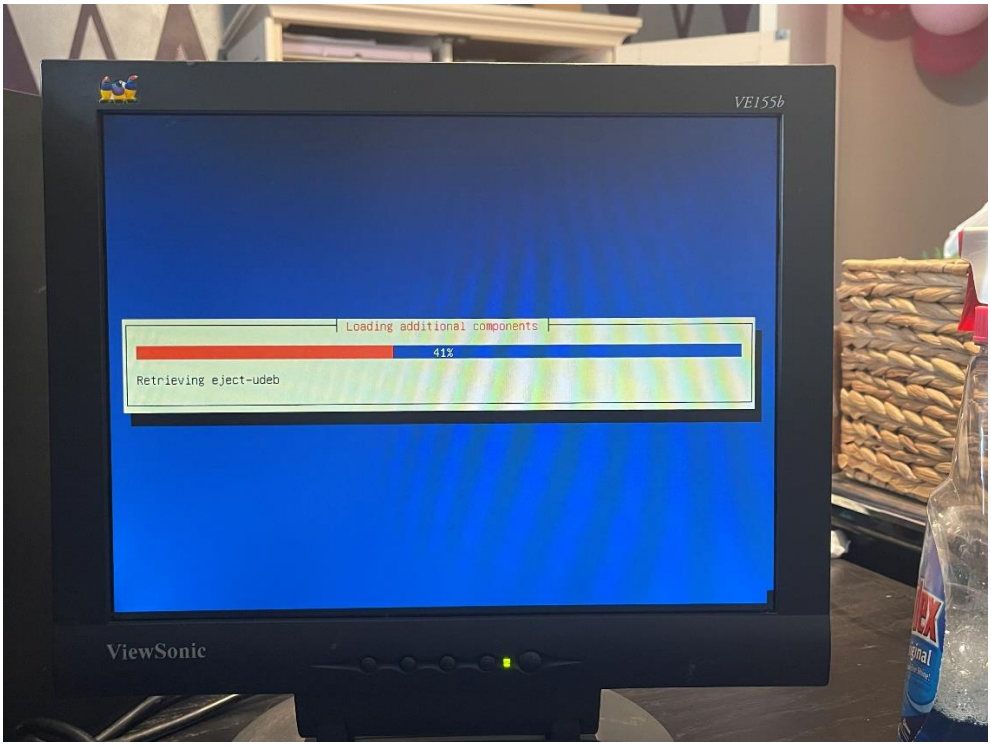
Tell Debian where you live. This is the United States for me.



Select your keymap – American English is what I used.



The installer will now load additional components. Good opportunity to clean your screen if you want.



Non-free device firmware prompt

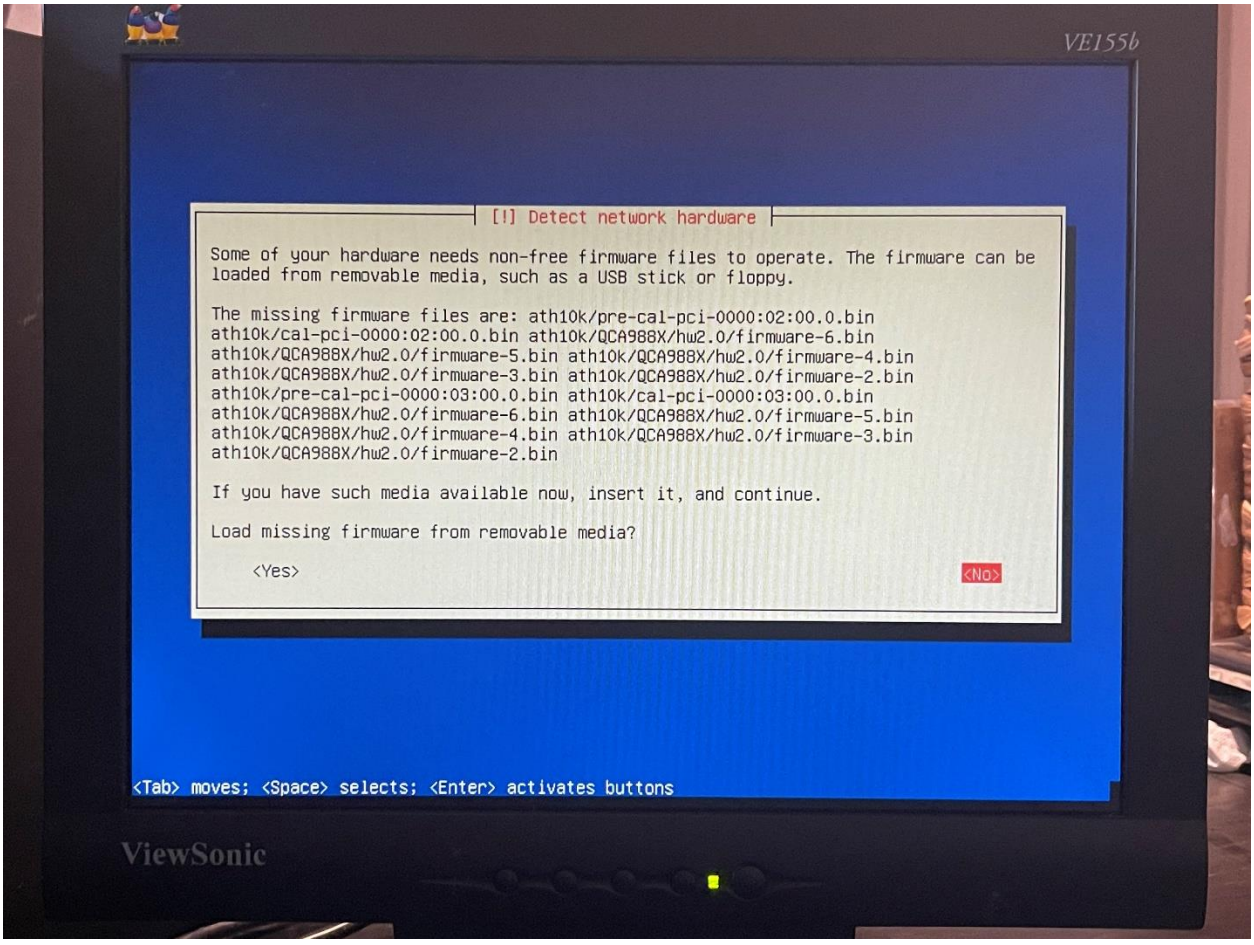
Often, Wi-Fi and some network cards will need device firmware loaded on them.

Since device manufacturers don't release their device firmwares under redistributable and modifiable FOSS terms, it's not included with Debian. It's easy to get after install, so no worries – unless of course you need the network card to install additional packages.

My wired network card is not in this list so I can proceed. So I'm answering No.

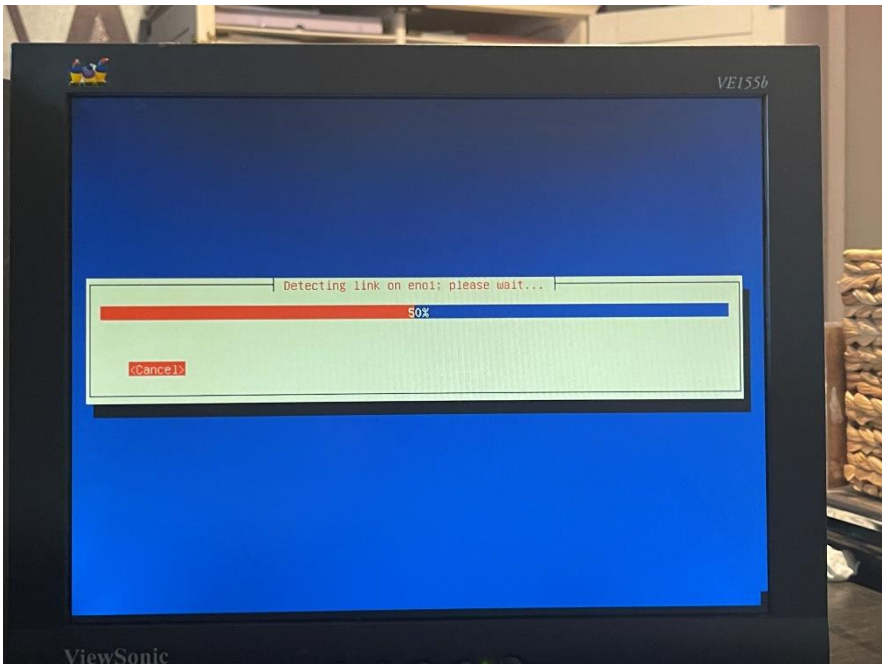
(If you were installing on a system that only had Wi-Fi and wanted to connect the installer to the Internet to get updated packages – you would need to get these firmware files now!)

You're seeing a bunch here because this system had 2 Wi-Fi cards in it – I was trying to build my own wireless access point. I did have the wired Ethernet interface connected though.

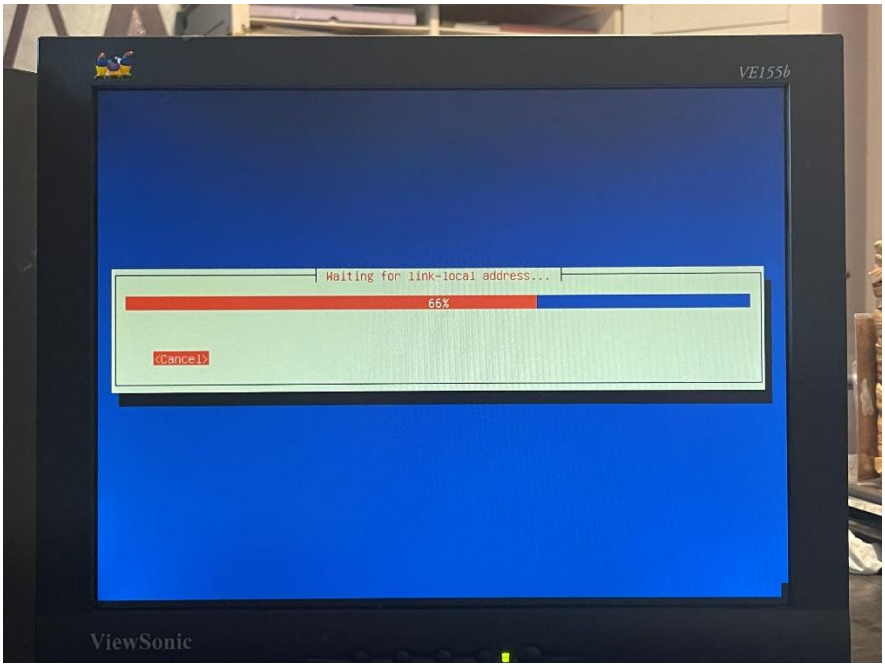


Installer Internet Connection

The installer will try to get an IP address on any network adapters it finds. I believe it starts with wired ones if they are there.

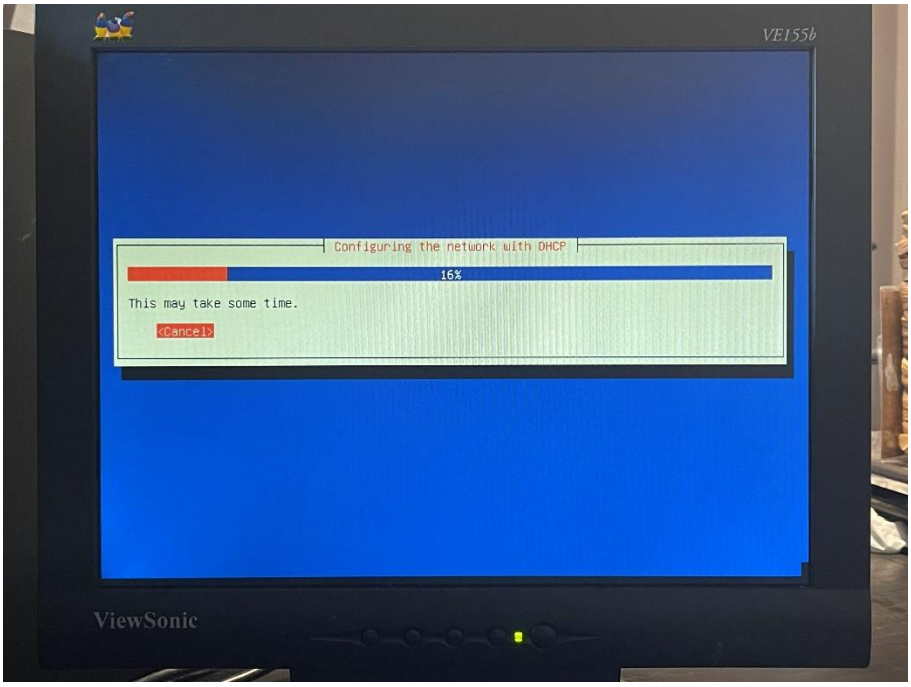


“Link-local address” is an IPv6 thing.

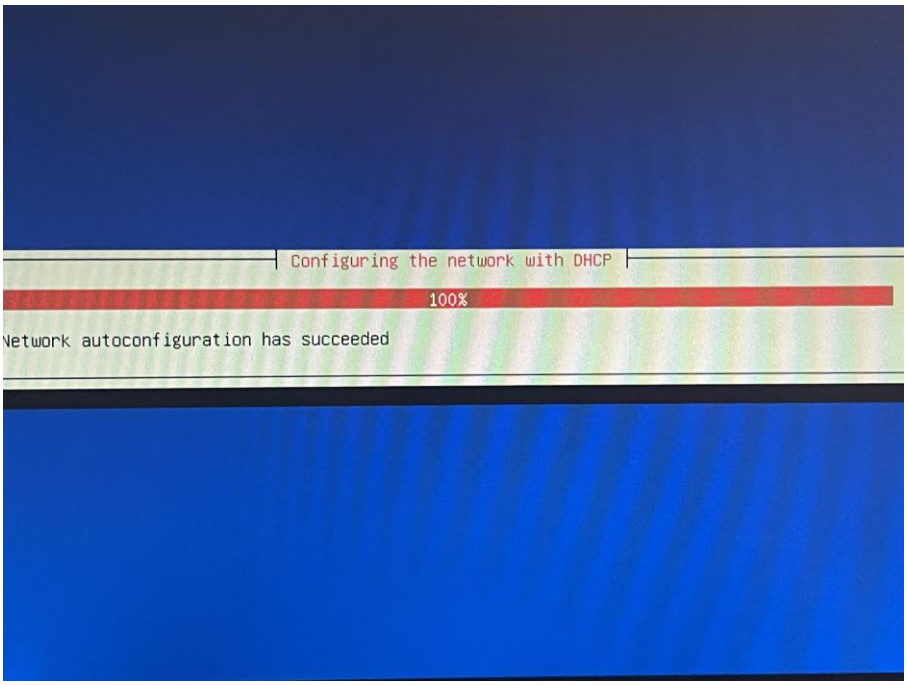


- If you were to install on a system with working Wi-Fi adapters, you would see prompts asking for an SSID (network name) and password at this point.

If the installer detects a link (connected cable), it will use DHCP to get an IP address.

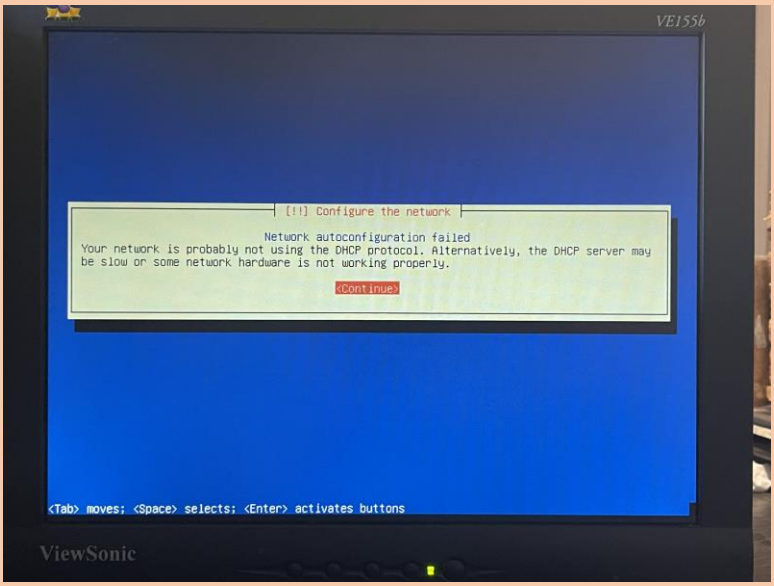


If all goes well, you'll see this:

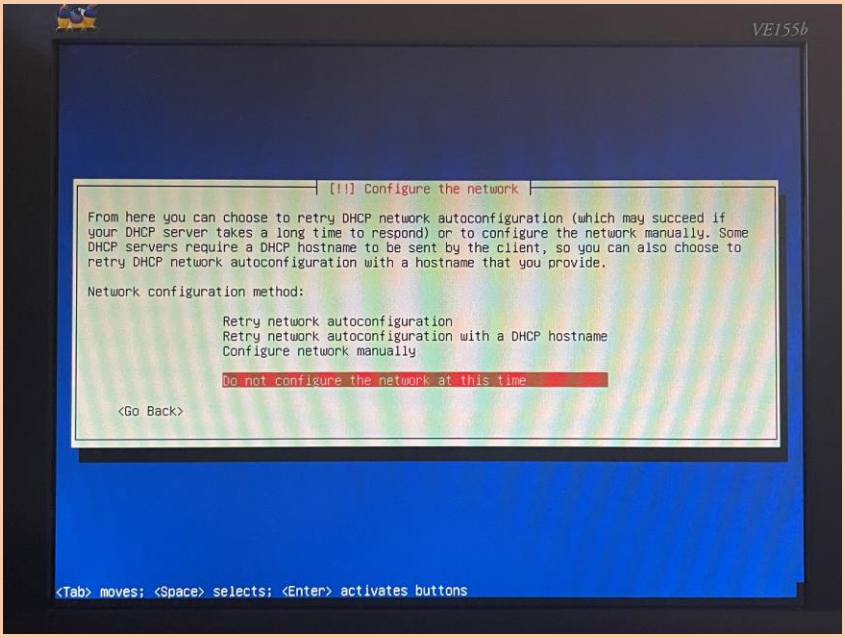


Network autoconfiguration failed

If the installer can't get the network working (or you cancel it), you'll see this:



Hitting "Continue" takes you here:

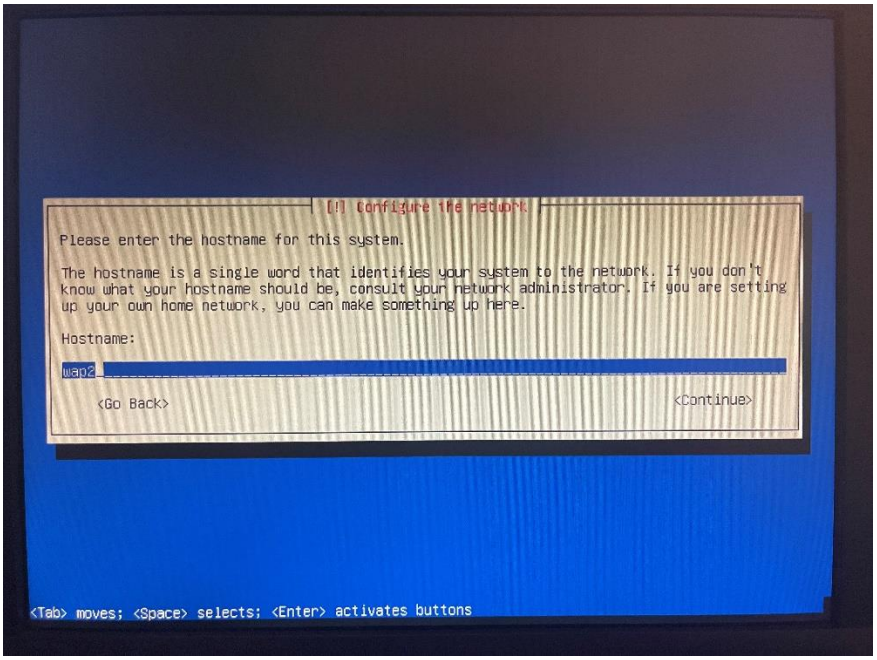


From here you can retry, configure manually (static IP, subnet, manual DNS), or leave it unconfigured.

Host and domain name

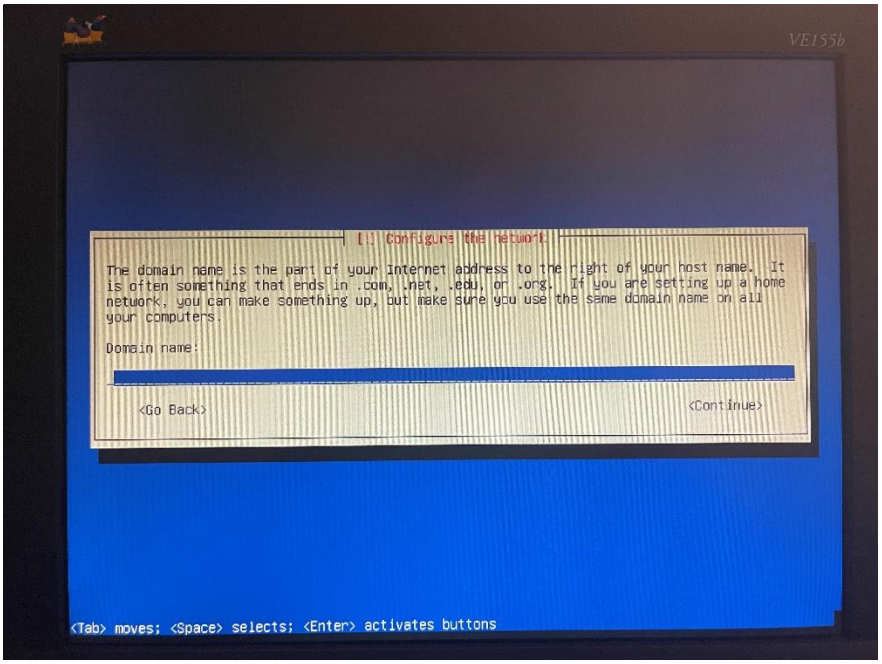
The installer will ask you for a hostname at this point.

The default is fine, but if you have more than one system, you should choose something to help you know what system you are on if you are remoted into it. It can be changed later.



The installer will ask you for a domain name next.

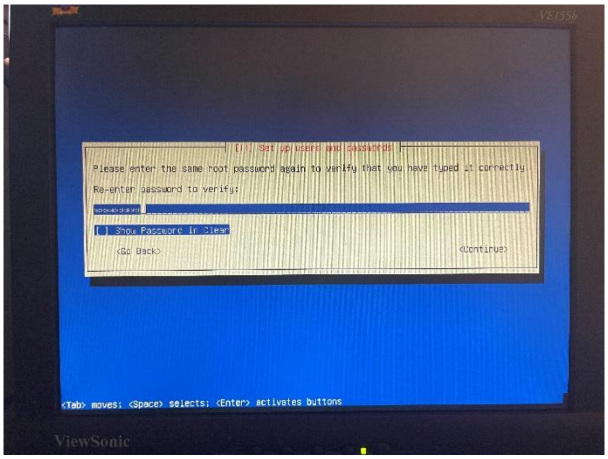
You can just hit Enter here unless you have some sort of DNS that will resolve to the IP address of this computer (if you don't know what this is, then you don't). This also can be changed later.

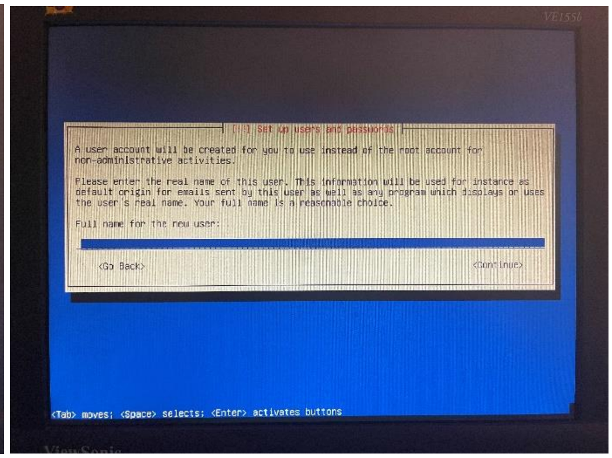


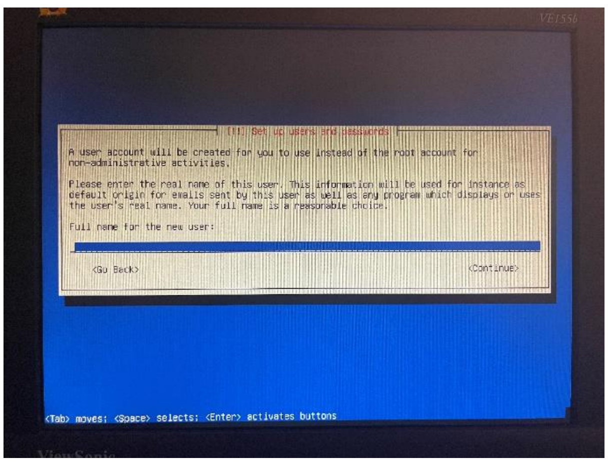
Local accounts setup

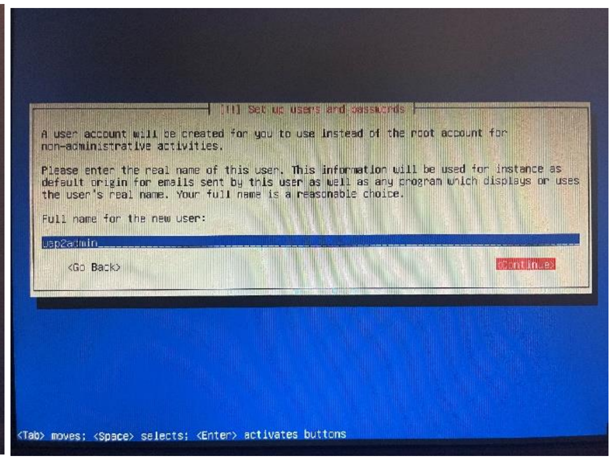
The installer will now ask you for:

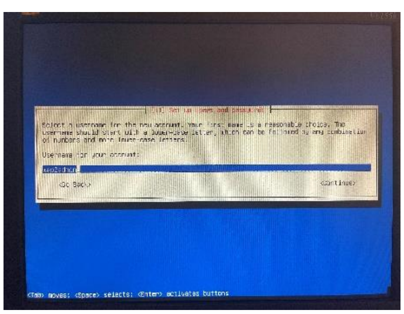
- a root password (to set)
- what name you'd like to give a non-root account
- a password for that non-root account

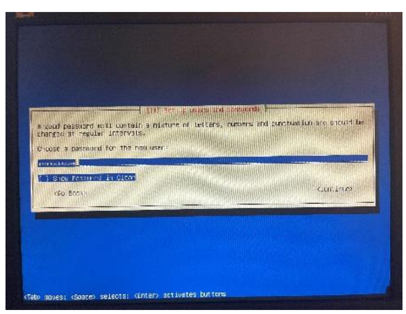


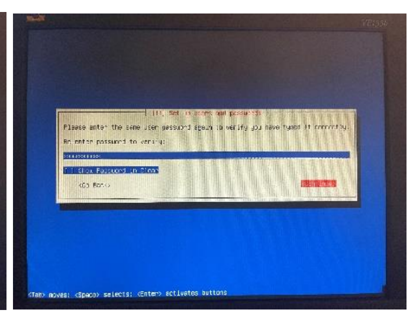








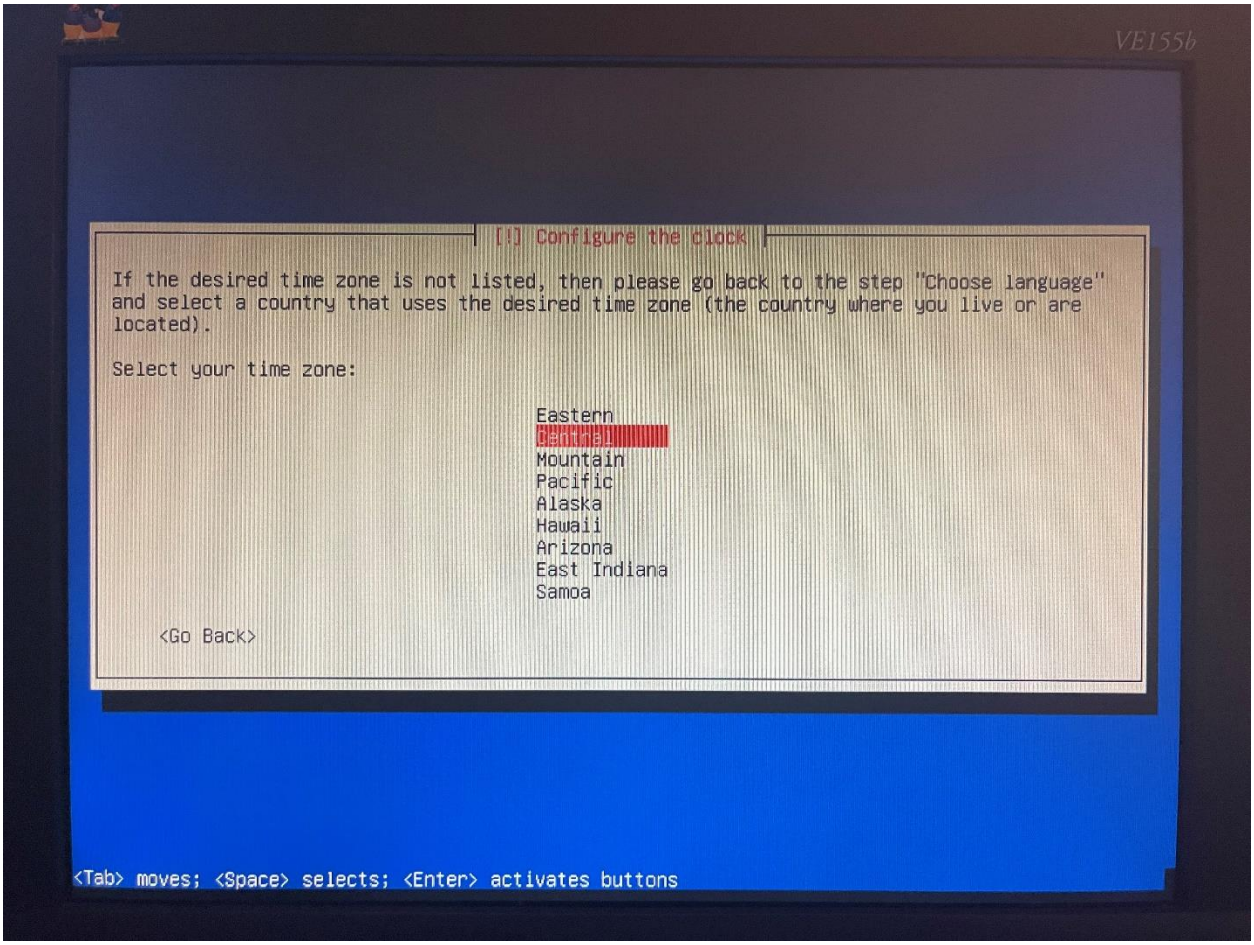




Time Zone

Linux keeps time using UTC so setting the timezone correctly is important for the system time to be correct.

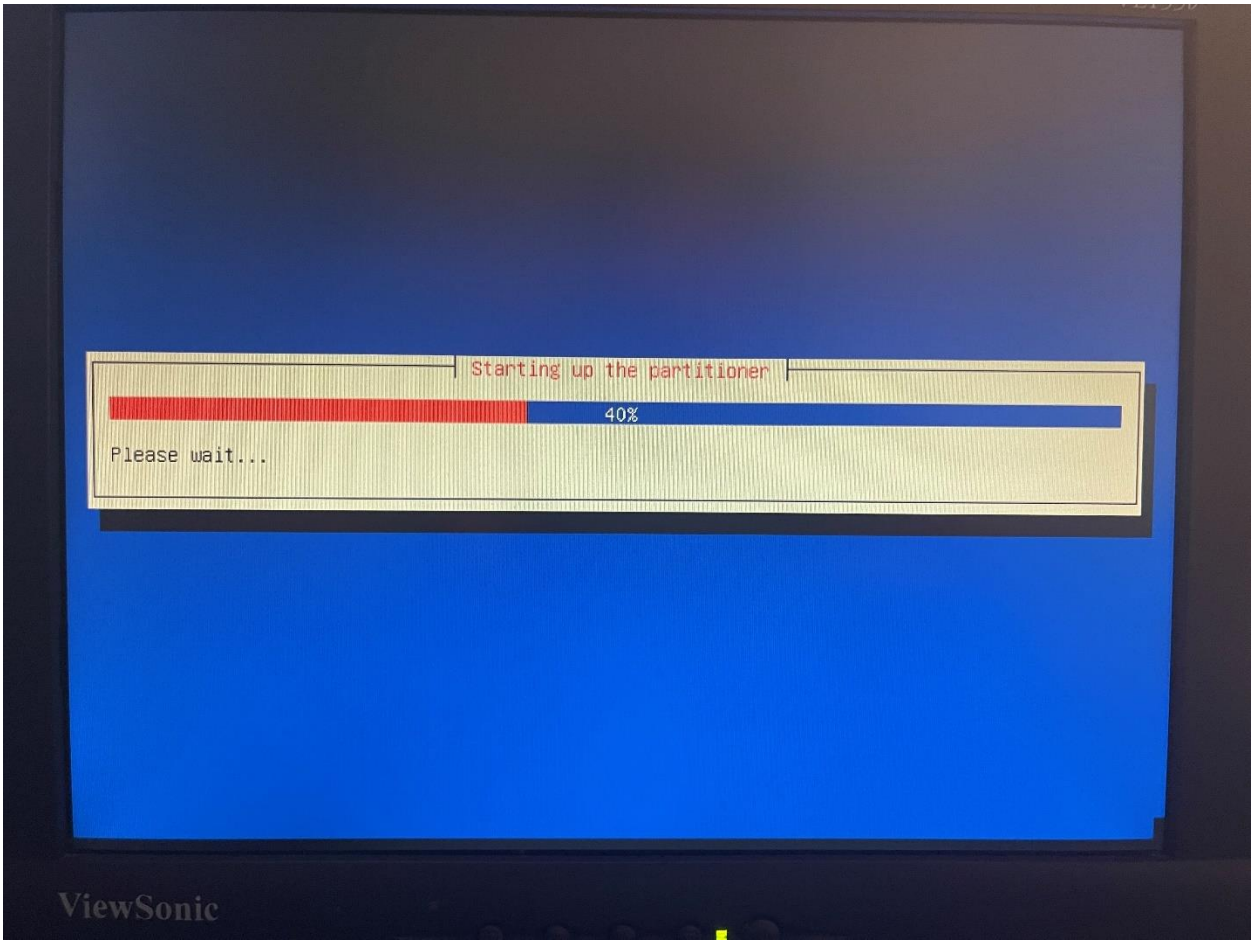
- Correct system time is important if you do anything with logs or certificates.
- HTTPS uses certificates so if your system time is wrong, web stuff may start acting weird or not working.



Partitioner Time!

All right, this is where things get FUN.

The partitioner will take a moment to start up. You will also see this screen flash by when it updates the list of volumes.



How Storage Devices Work Under Linux

We need to talk about how Linux addresses storage devices (disks, drives) and some features Linux has in regard to them.

Physically connected storage devices and device files.

We begin with storage devices that are physically connected to the system. Linux detects these and creates *block device files* for them in `/dev`.

- Any block device file in `/dev` is basically a “disk” or “drive”, or something that acts just like it:
    - Block device files can be formatted.
    - Once formatted, they can be mounted, and the storage used to hold files.
- Linux obeys the classical UNIX “everything is a file” philosophy – so storage devices get their own block device “files” in `/dev`.
    - Really these are “device nodes”, not files, but programs can work with them just like files.

    - Only root has direct access to them.
    - For the potentially reckless: Don't ever change permissions of these files without knowing what you are doing.
- For the curious: *Block devices* are distinct from *character devices* which includes things like serial ports and your keyboard. (Those are in `/dev` too).

BIOS/UEFI partitions

As you will learn, not every block device file in `/dev` represents a physical storage device – but physical storage devices should definitely have a file in `/dev` (if they're working).

Storage devices might have a BIOS (MBR) or UEFI (GPT) partition table.

- For the curious: All that a partition table really is a small bit of data at the beginning of the storage device.

If Linux detects a storage device with a partition table, it will create individual block device files for each partition.

- So if your disk has 3 partitions, Linux will create the block device files `/dev/sda1`, `/dev/sda2`, and `/dev/sda3`, for example.
- Each of those are basically separate "disks" or "drives" to Linux.
- These partitions are "in" `/dev/sda` which is still there.
  You do not want to use or really think about `/dev/sda` anymore except in very specific circumstances – think of it as representing the whole physical device. If you write to `/dev/sda` you may overwrite data that lives in `/dev/sda1`, `/dev/sda2`, or `/dev/sda3` within in. Most programs will not let you do that, but some will.

Virgin Storage Devices

If you attach a new SSD, for example, fresh from the factory, it won't have an MBR or GPT. All you will see is, again for example, `/dev/sda`.

To create partitions, I like `cfdisk`. **You should always at least create 1 partition, especially on drives that will be involved in booting.** Partitioning creates space for the bootloader (which isn't in any partitions), and PC firmware expects partitions.

MD: Software RAID (`/dev/md` device files)

Linux has a feature called MD which enables "Software RAID".

- You can tell Linux to take two block device files and make them work like a RAID-1. For example, `/dev/sda1` and `/dev/sdb1`. The block device files have to be similar sizes.
    - Linux absolutely supports things like RAID-5, RAID-10 and more.
- After it is setup, you will get another block device file - typically `/dev/md0` (or another number).
- `/dev/md0` is "on top of" `/dev/sda1` and `/dev/sdb1` which is still there.
  You do not want to use `/dev/sda1` or `/dev/sdb1` any more. They are now "owned" by `/dev/md0`. If you write to `/dev/sda1` or `/dev/sdb1` you may overwrite parts of `/dev/md0`. Most programs will not let you do that, though.

- Reminder: RAID-1 means two drives have two copies of data, and if one dies, the system automatically uses the other copy.
- So RAID-1'ing 2 128GB drives gives you a total storage capacity of 128GB, well protected.
    - Yes, you can do something silly like make a RAID-1 with two partitions on the same physical device (e.g. `/dev/sda1` and `/dev/sda2`). At least I think it will let you (to be honest I haven't ever tried). If it would work, don't do that, that's dumb. You want the partitions on DIFFERENT physical devices.

LVM: Volume Groups And Logical Volumes

Linux has a feature called LVM which provides the features known as Volume Groups and Logical Volumes.

- A volume group (VG) is a collection of one or more block device files
    - For example, you might make a volume group out of `/dev/sda1` and `/dev/sdb1`.
    - The volume group will have a name, such as `examplevg`
    - `examplevg` is "on top of" `/dev/sda1` and `/dev/sdb1` which is still there.
      You do not want to use `/dev/sda1` or `/dev/sdb1` any more. They are now "owned" by the VG `examplevg`. If you write to `/dev/sda1` or `/dev/sdb1` you can damage the VG or any LVs in it (keep reading). Most programs won't let you do that, though.
    - Let's say `/dev/sda1` is 200GB and `/dev/sdb1` is 500GB. examplevg therefore is 700GB.
- A Logical Volume (LV) is simply a portion of that volume group – exposed as another device file.
    - For example, you might make an LV from `examplevg` that's size 300GB named data
    - This will now be a new block device file `/dev/examplevg/data`.
    - Storage in `examplevg` isn't accessible until you expose some or all of it in an LV.
- You might be thinking LVs are like partitions that have the option to span multiple device files.
    - You are right.
    - LVs have additional capabilities too:
        - They can be resized anytime, as long as there's enough room in the VG.
        - You can add more block device files to the VG any time, or remove them if no LV is using them.
        - You can do snapshots – back up a partition while it's in use. There needs to be enough space in the VG to hold data written while the snapshot is being held until you're done.
- **The snapshot capability one reason why you would want to use LVs even if you don't plan to combine the storage of multiple block device files.**
- **You can put a `/dev/md0` (RAID) in a VG – Linux is fine with it.**

LUKS - Encryption

Linux has a feature called LUKS which is encryption.

- You can tell Linux to encrypt a block device file.
- For example, you might want to encrypt `/dev/md0` (could be any block device file)
- LUKS needs a key (password) and a name. Let's say you choose "`md0encrypted`" for the name.

- Once unlocked, there will be a new block device `/dev/mapper/md0encrypted` – readable and writeable like normal.
- Behind the scenes writes to go `/dev/md0` encrypted.
- When you lock or close `/dev/mapper/md0encrypted`, it disappears, and `/dev/md0` only contains gibberish.
- **<u>You do not want to use `/dev/md0` directly any more.</u>**
  It is encrypted, if accessed directly and not through the `/dev/mapper` device node, you'll bulldoze your data. Some programs can detect the presence of LUKS and will not let you do this.
- **Linux is flexible – encrypted block device files can be in VGs, and any block device file can be encrypted, including LVs.**

Swap partitions

If Linux runs out of RAM due to processes/apps requesting too much, it can *swap* parts of RAM to a storage device. Storage devices (even SSDs) are much slower than RAM, but this prevents programs or the system from crashing.

It also may to swap stuff that hasn't been used too recently to RAM so more free RAM is available for random things. Things get called back in to RAM automatically as needed.

It's always good to have some swap just in case, unless you 100% know for sure your RAM usage won't ever exceed your total RAM. That's hard to do, and setting up swap is easy, so I usually set up swap. (If your server has more RAM than storage, then by all means don't set up swap.)

As RAM can contain things like encryption keys, swap should be encrypted. Debian supports encrypting swap with a unique key on each boot – meaning nothing from the previous boot can be recovered from it—as that unique key is never saved anywhere.

The Plan

We have two 128 SSDs, and will do the following with them using the Debian installer partitioner.

- Step 1: Create 1 GPT partition on each SSD that's 96GB.
  Allows booting off of either disk
  We'll leave the other 32GB on each SSD free for other use.

- Step 2: Create an MD RAID-1 using 2 block device files:
  1) he single GPT partition on the first SSD
  2) the single GPT partition on the second SSD

  Now we'll have a single 96GB hunk of storage protected by RAID-1.

- Step 3: Create an LVM VG containing 1 block device file: the RAID-1 we created in step 2.
  Now we can use LVs.

- Step 4: Create several LVs in our VG for various Linux needs:
  - `swap`
  - `/` (root)
  - `/tmp`

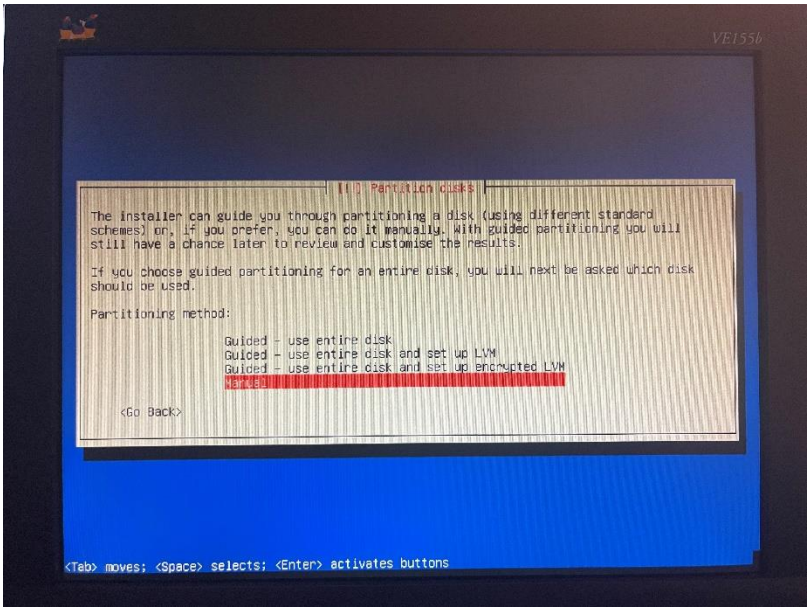
    - /var
    - /home

- Step 5: Assign mount points/purposes to each LV.

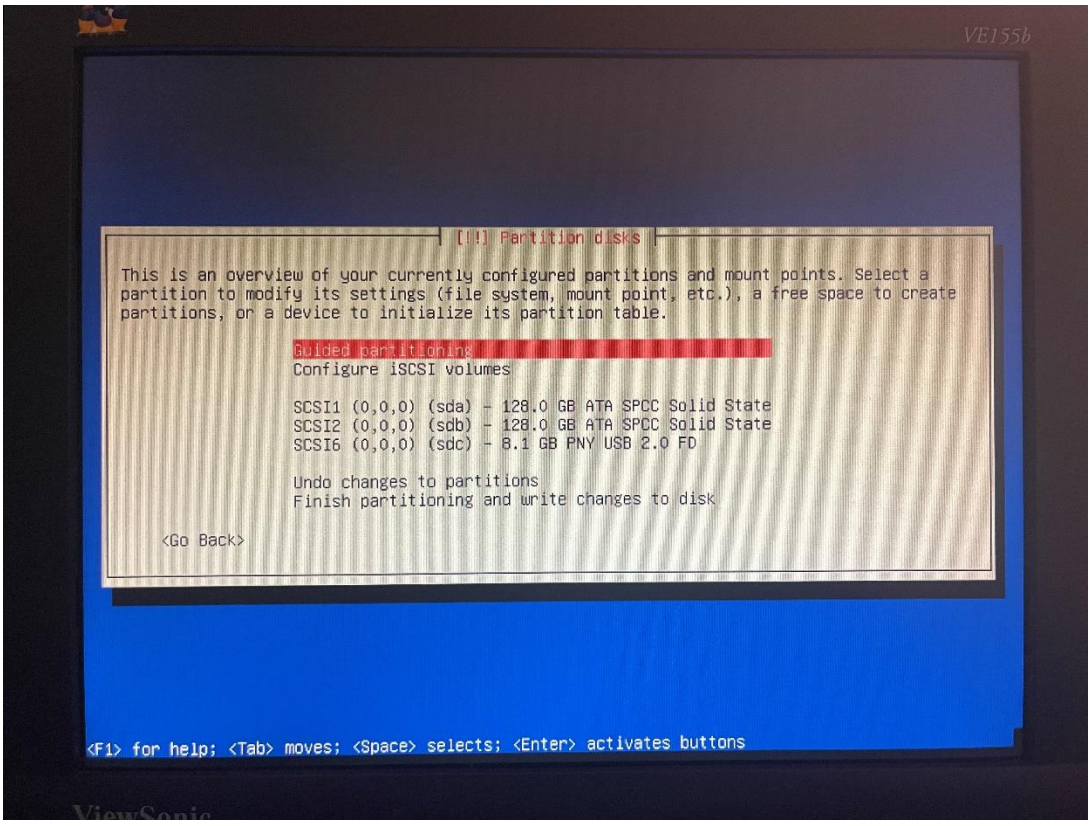
- Step 6: Encrypted swap setup

Executing The Plan

We're doing it manually, so we'll select "Manual" at the bottom.



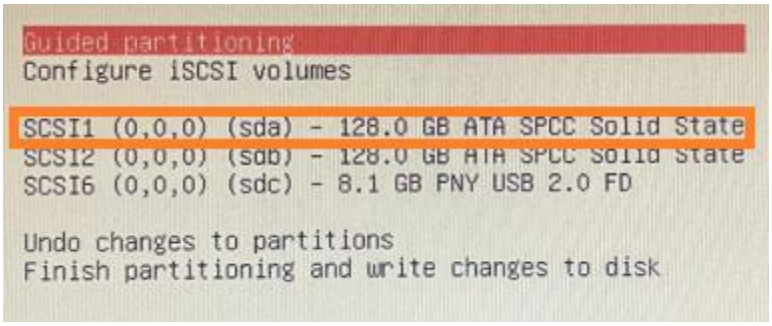
The next screen lists devices, partitions (block device files), and mount points. The screenshot below shows 2 virgin fresh-out-of-the-box "128.0 GB ATA SPCC Solid State" SSDs, and they do not have any partition info on them yet. The third "8.1 GB PNY USB 2.0 FD" drive is the USB drive being used to install this – you shouldn't mess with that while in the installer!



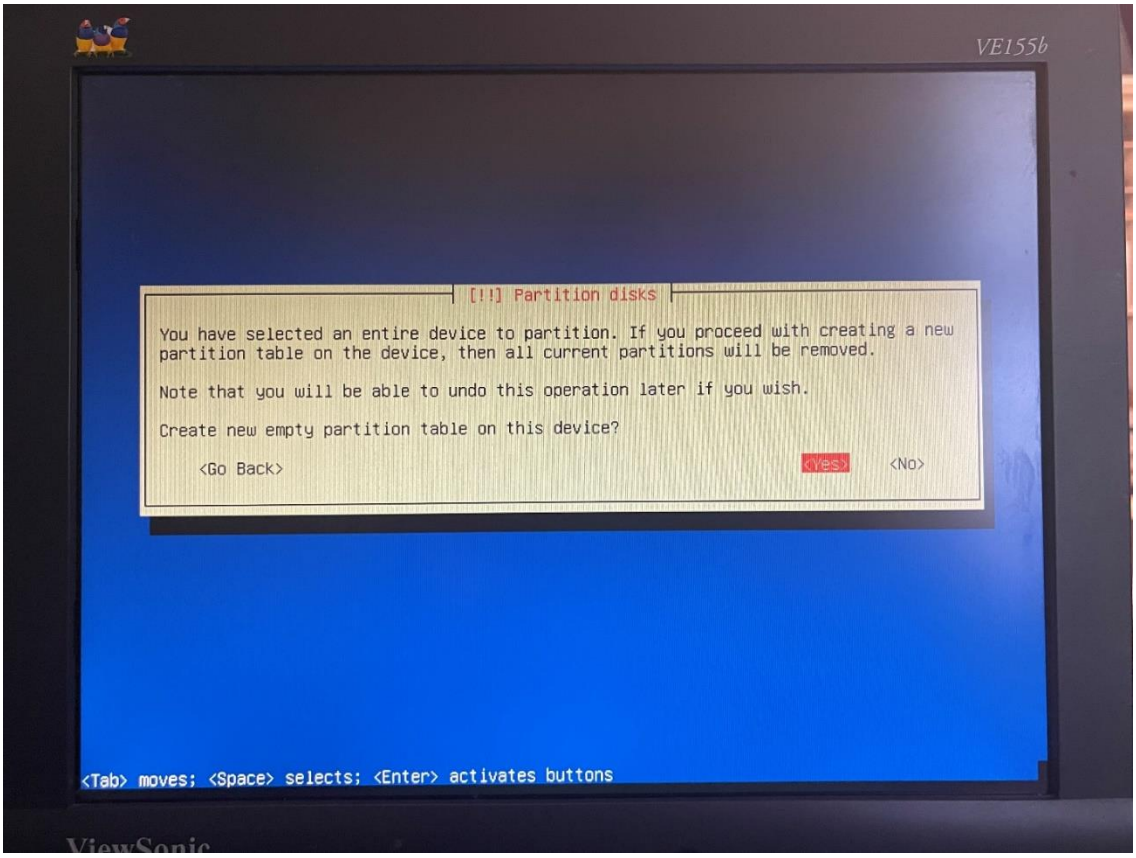
Step 1: Create 1 96GB BIOS MBR partition on each SSD (sda and sdb)

Let's get some GPT partitions on these puppies.

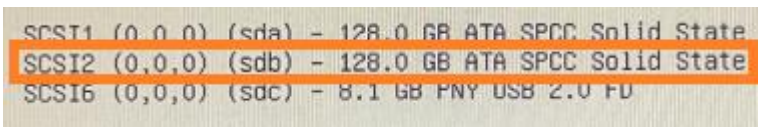
Move the highlight to the first drive (SCSI1) and press Enter.



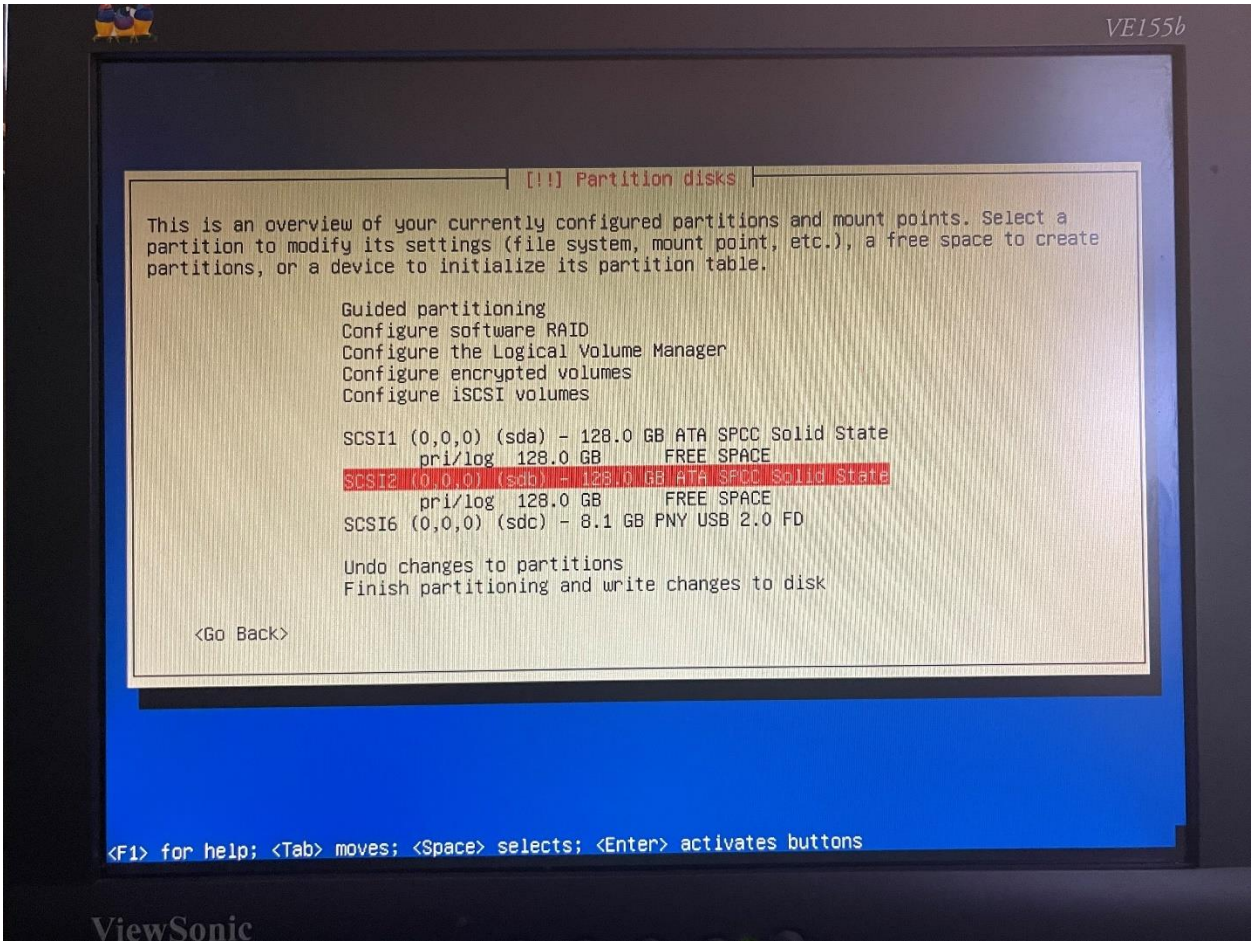
You'll see this, which is normal if you've never used the drive before or if it has a totally blank MBR/GPT partition table.



We want to do this for both drives SCSI1 and SCSI2 - simply select them from the menu, press Enter to bring up the above, and say Yes to this dialog. Do the same with SCSI2:

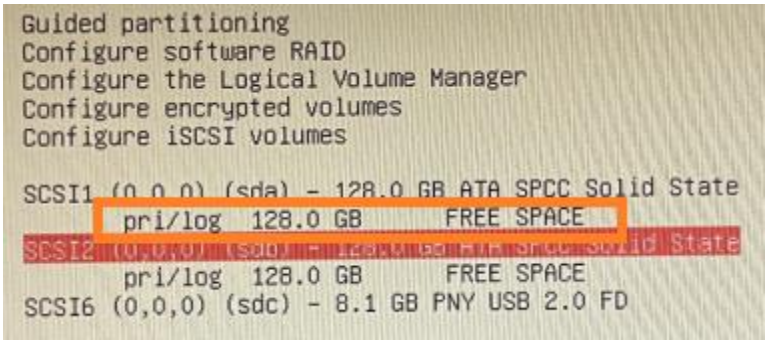


When we've done that, now you can see FREE SPACE beneath both drives.

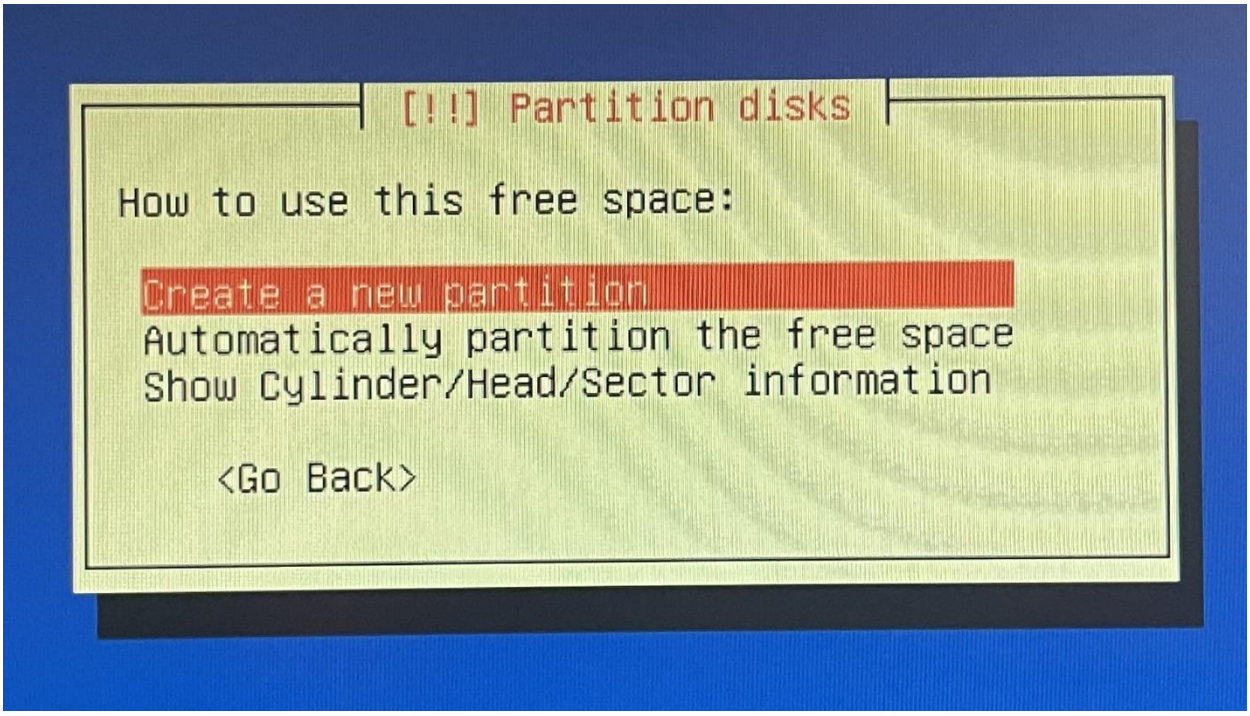


Now we want to actually make a partition in both those free space. We'll start with SCSI1.

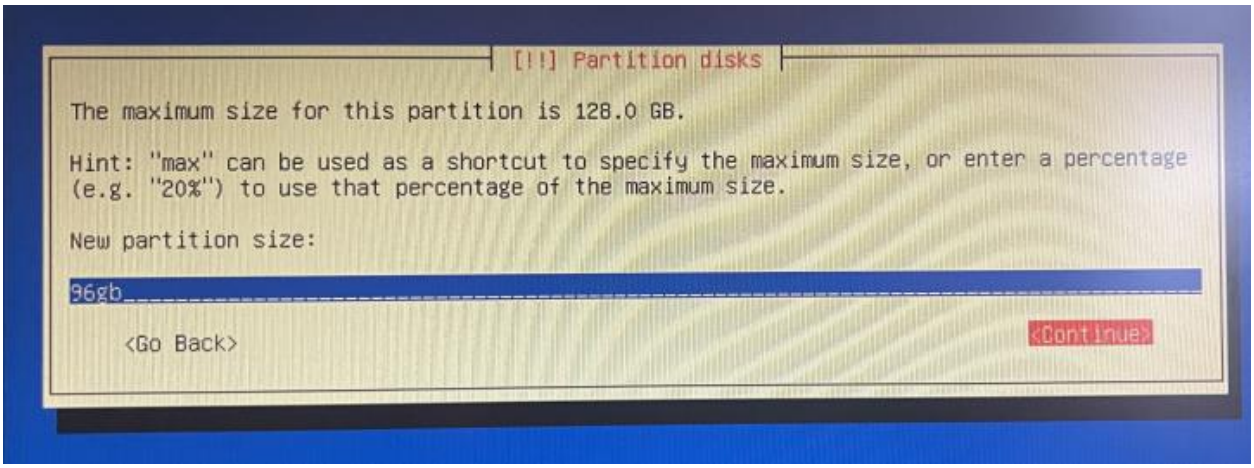
Move the highlight to SCSI1's FREE SPACE line and press Enter.



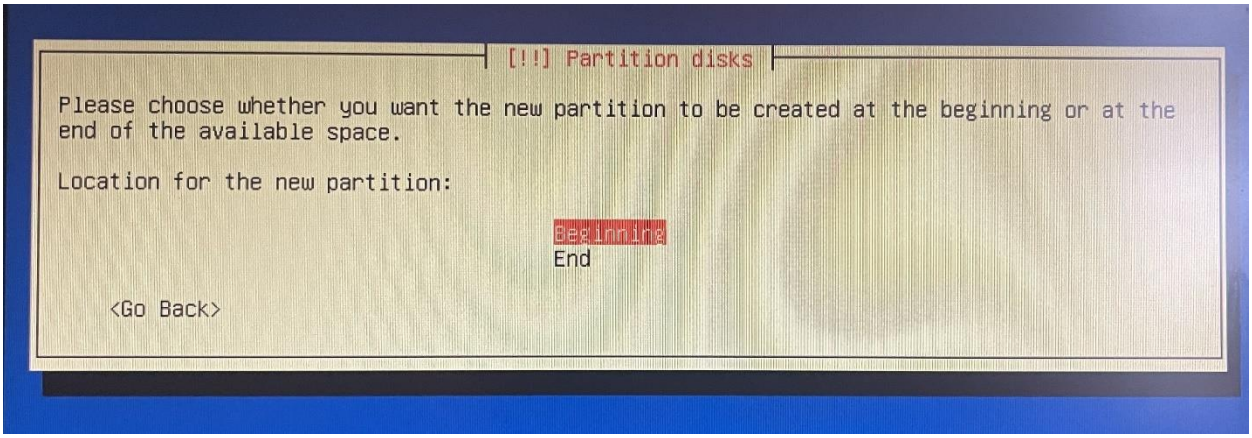
On this screen, say “Create a new partition”.



You’ll be asked to enter a size. Enter **96gb** and press Enter.

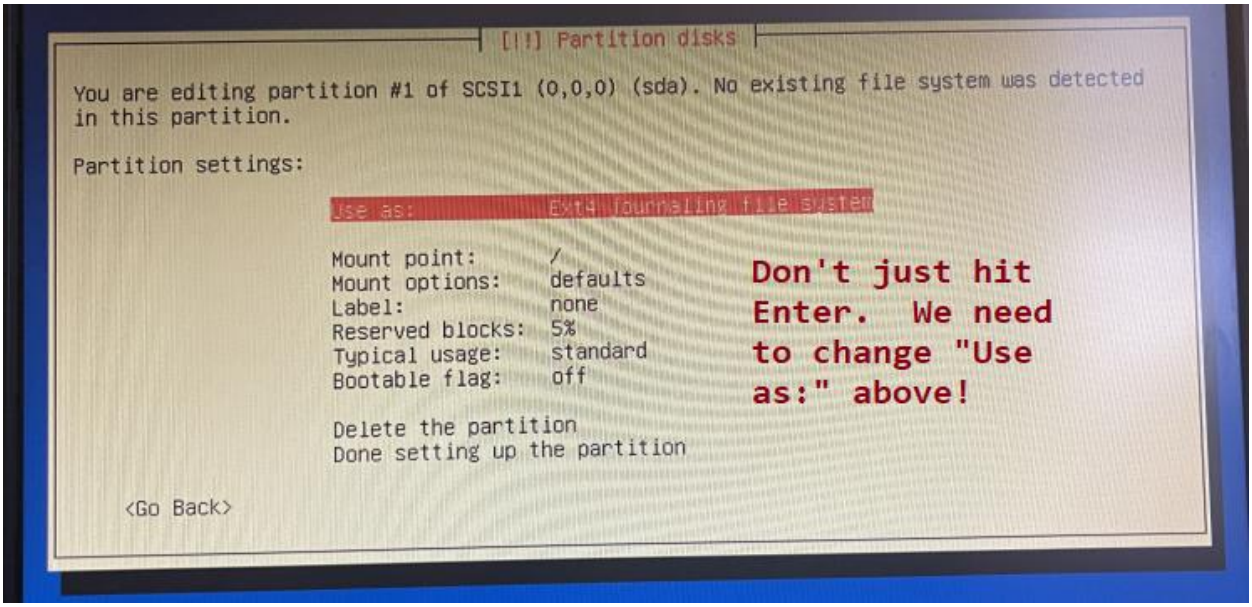


Say “Beginning” here.



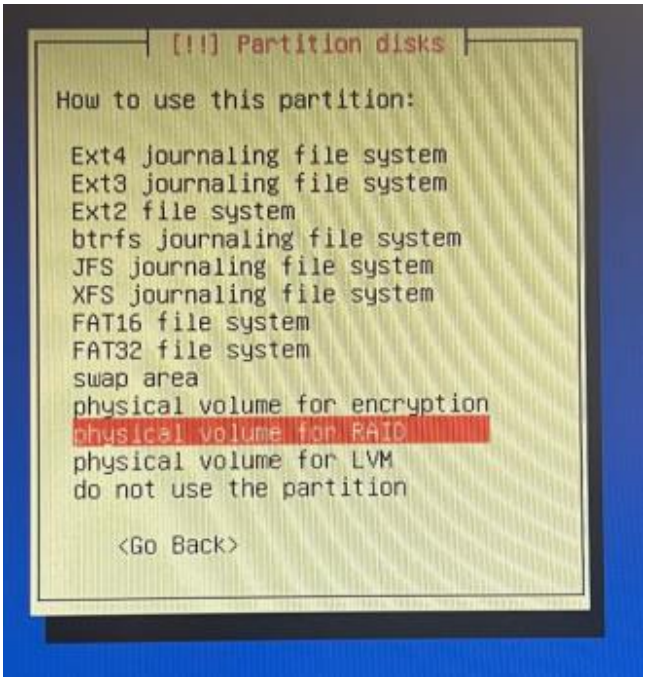
This is a screen you’ll see quite a bit – tells us what we want to do with the block device file (in this case, representing a GPT partition on a physical storage device).

We do NOT want to use it as “Ext4 journaling file system” – press Enter to bring up options.

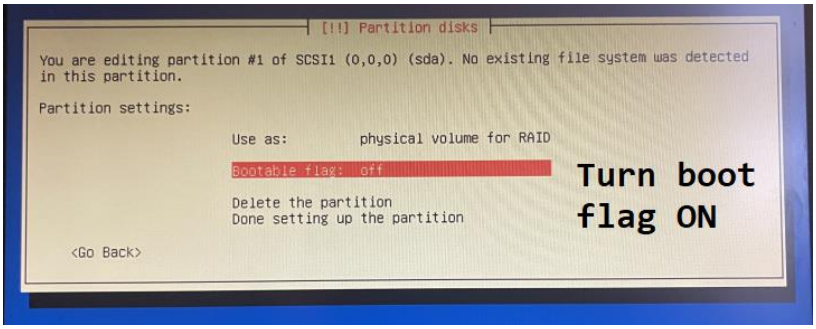


And you'll get a list of all the nifty things Debian knows how to setup on block device files.

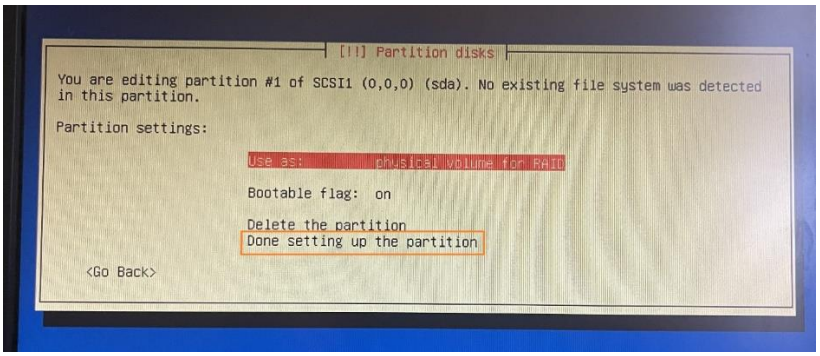
Move the highlight to "physical volume for RAID" and press Enter.



Then it brings you back here.



Highlight "bootable flag: off" and press Enter to flip it on – we do want to boot from these drives.

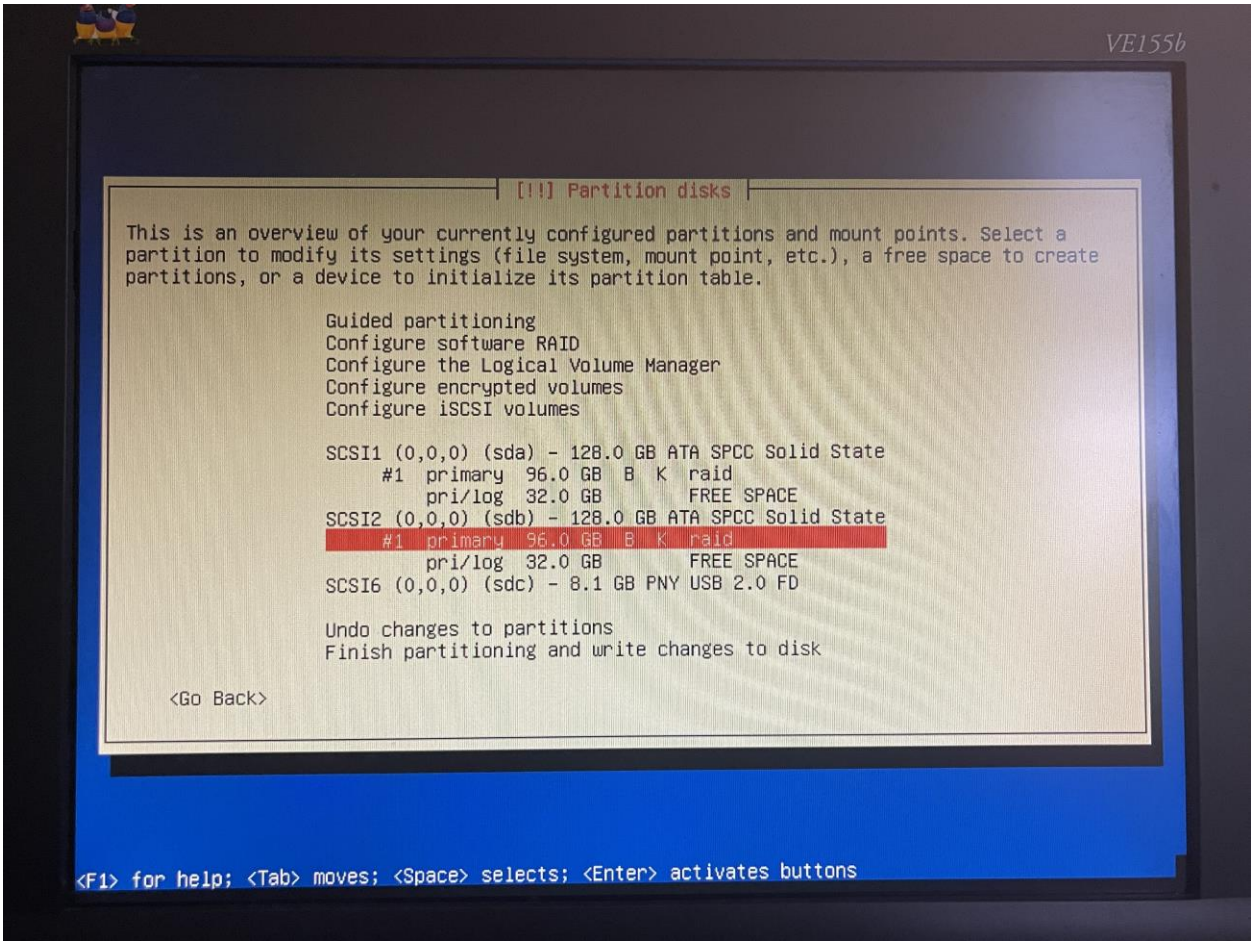


Looks good, move the highlight to "Done setting up the partition" and press Enter.

Next, you want to go to the other FREE SPACE line and repeat the above, same exact steps.

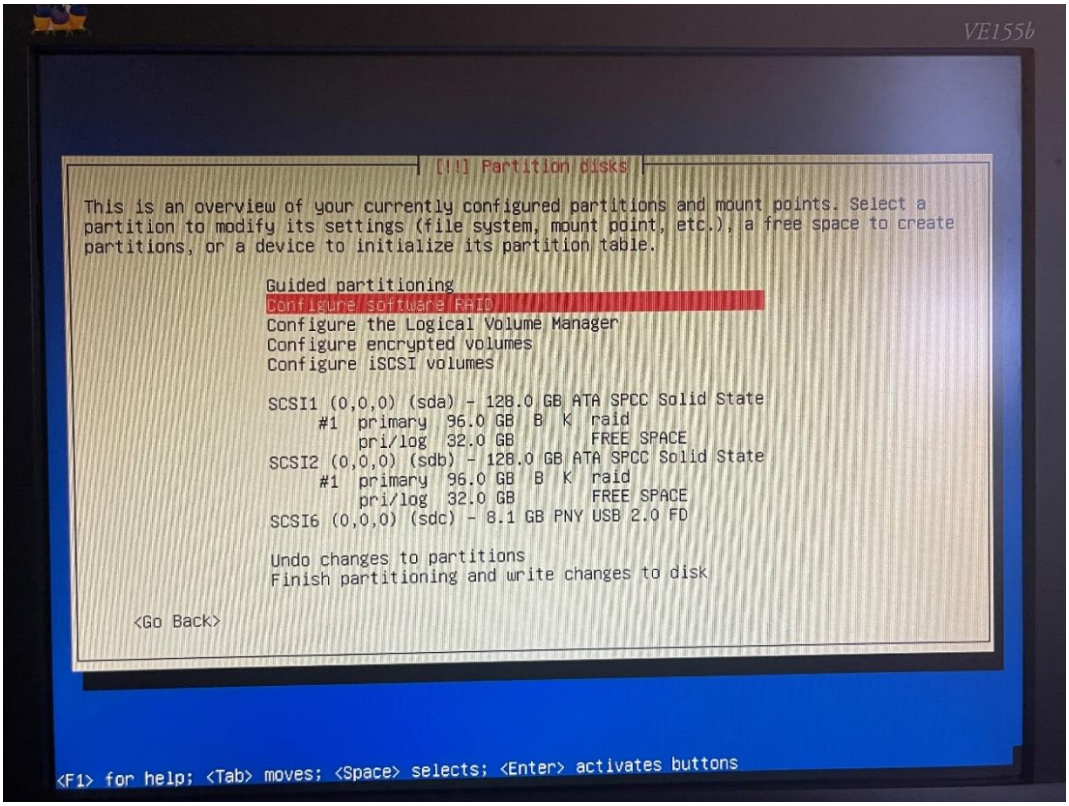
Make the second drive bootable too. If the first drive dies the system will boot off of the second one.

When that's done, things should look like this:



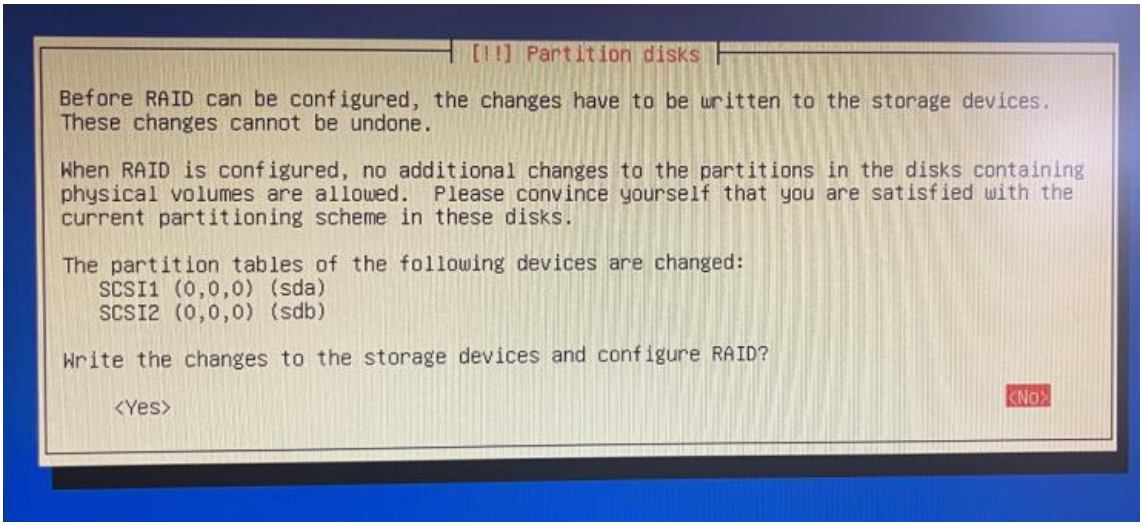
Step 2: Create an MD RAID-1 using the 2 new partitions we just created.

If you thought we were going to go up to the “Configure software RAID” option near the top, and press Enter, you would be correct. (If this is boring you to death and you are following it because you have to, you’re doing a great job. Keep going!)



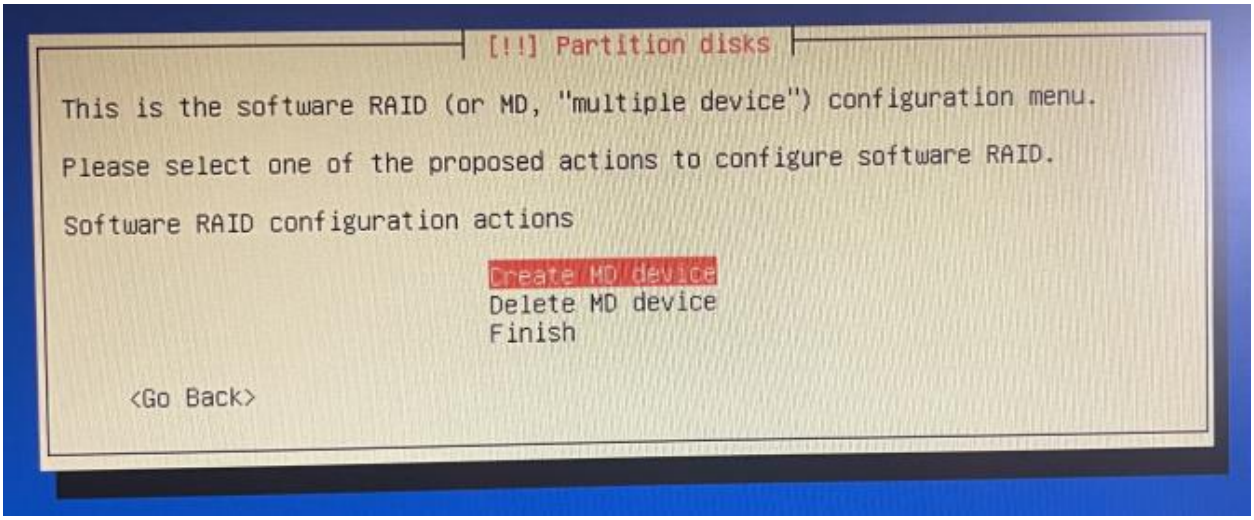
When you do that, you’ll see this warning.

It is saying it will actually write our previous partition changes to the storage devices. We want that here, so say Yes here. If you were using old drives, this would be your final warning to think about if the drives had something you wanted to keep.

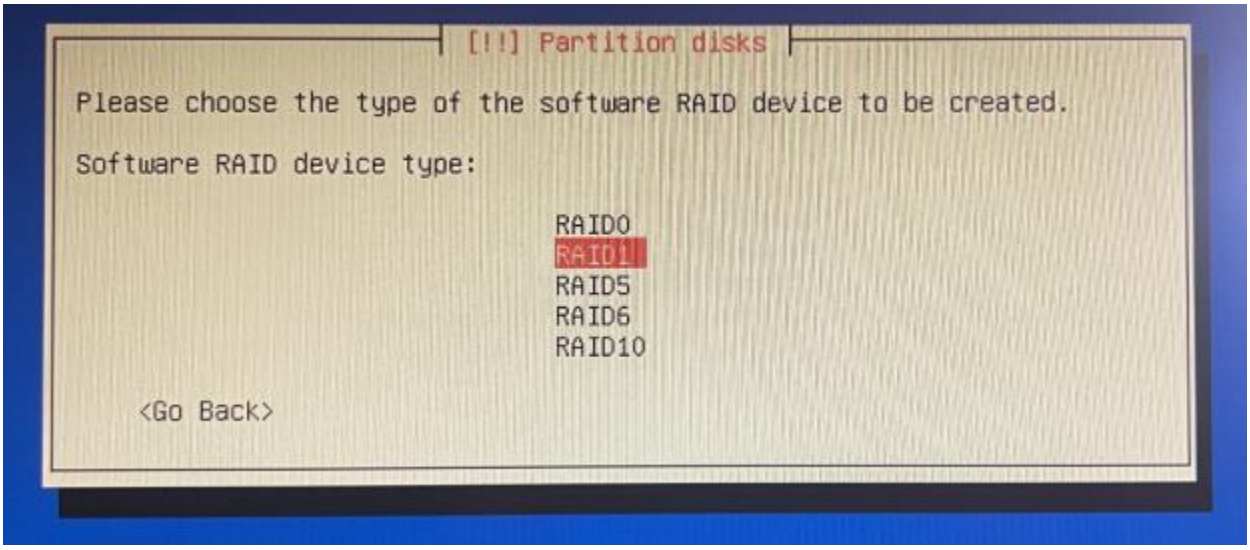


All right, now we're in the RAID menu.

We want to "Create MD device."

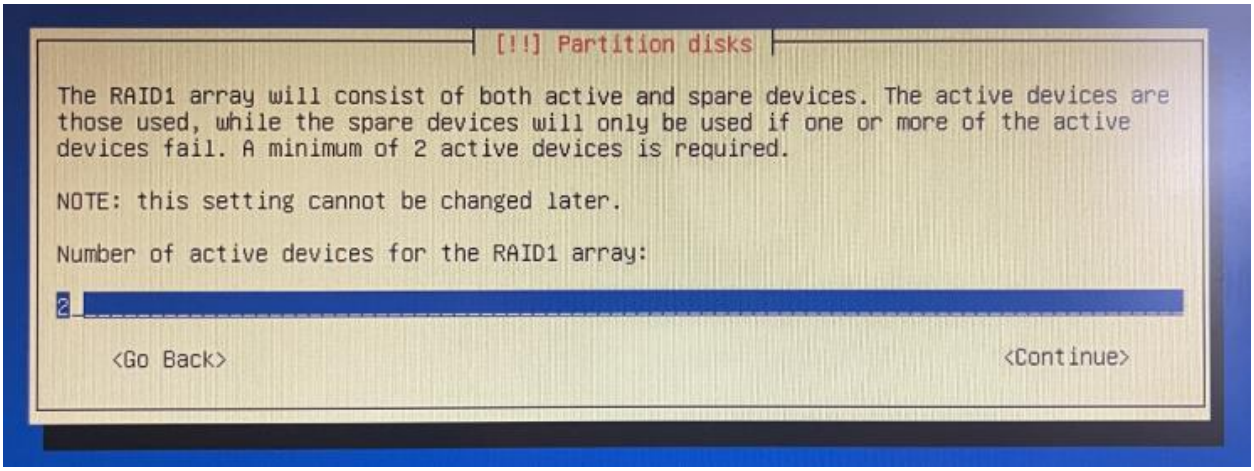


We're going to do a RAID1, so select that.



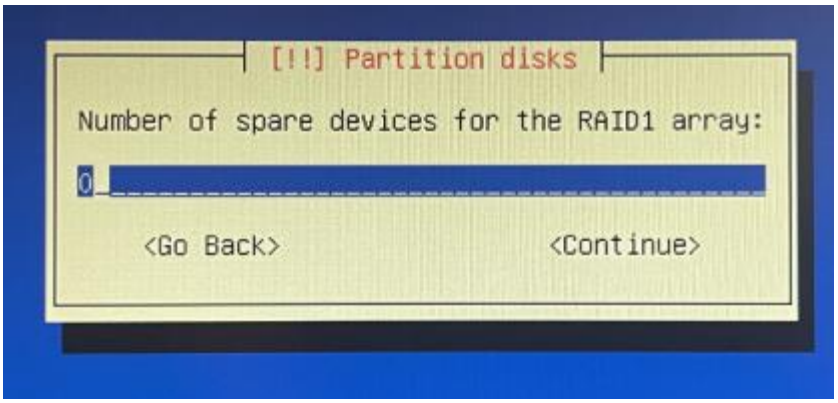
Now the installer will talk about active and spare devices, fairly straightforwardly.

Since we only have 2 SSDs we're wanting to RAID1 up, we'll put 2 here.



No spares, so we say 0 here.

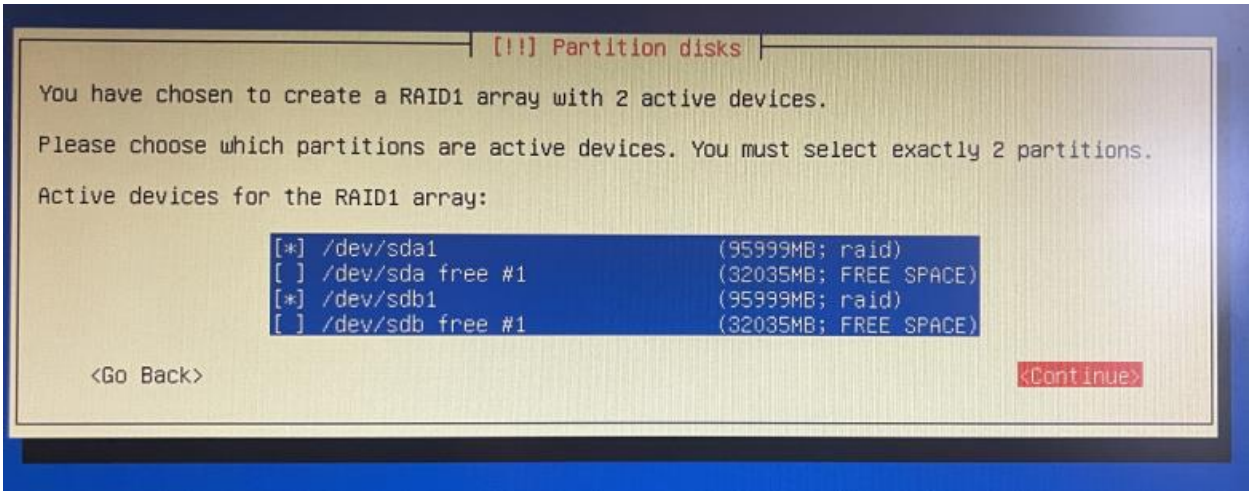
So if 1 out of 2 drives die, we are vulnerable until it is replaced. If we had installed more SSDs in this system, we could have brought them on as spares to this RAID-1 if we wanted to be extra paranoid about that.



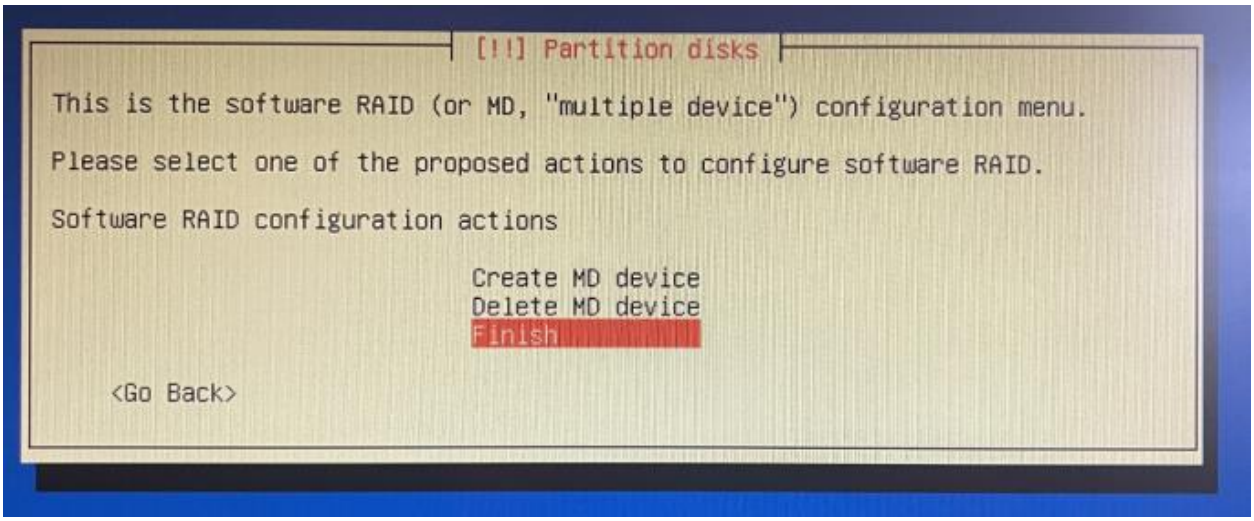
Now you get to tell it which block device files you want to put in the new RAID1 you're making.

The word "raid" here is the type of partition – that is set when you tell the installer "physical volume for RAID" above. It's NOT a real RAID1 yet.

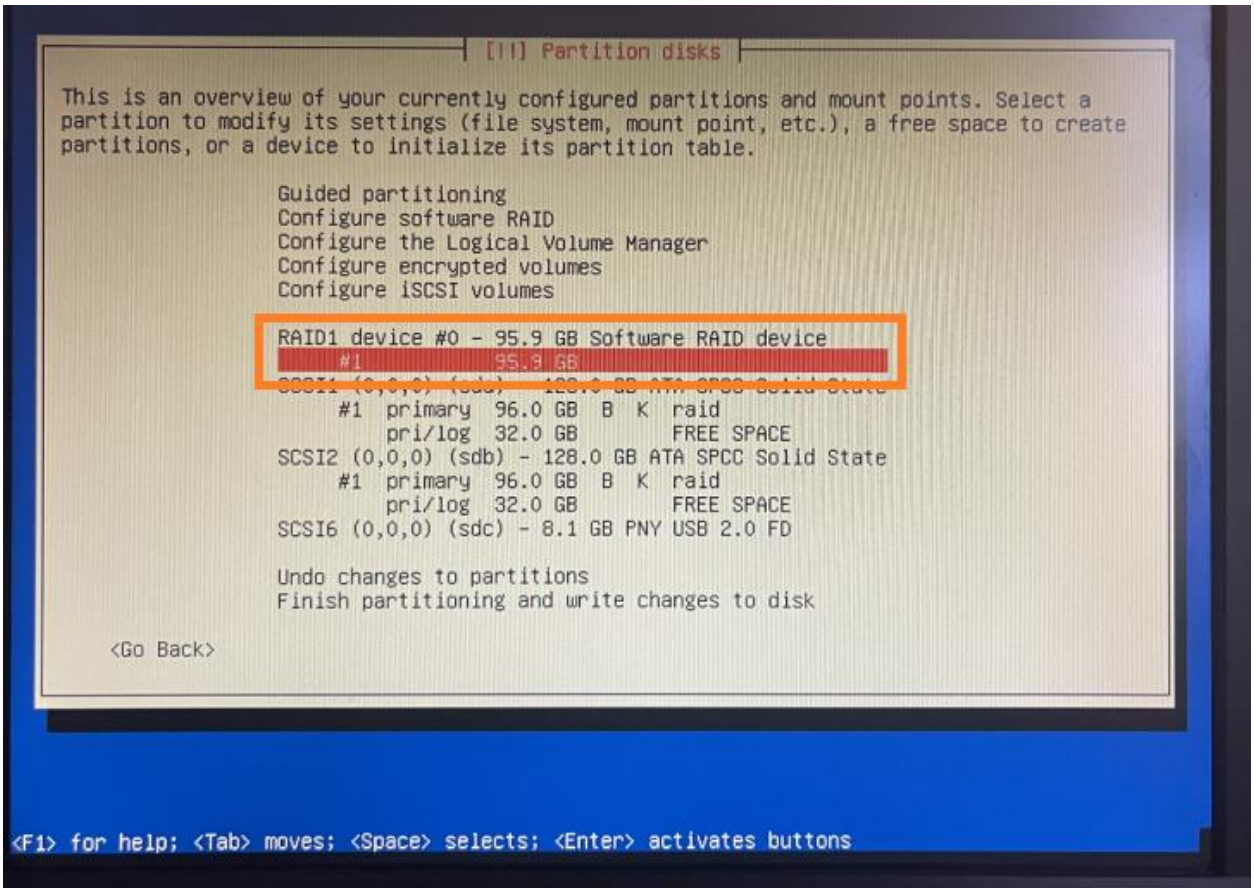
Don't pick the FREE SPACE ones here. Our plan is to RAID1 up sda1 and sdb1, so highlight and check those, then highlight and press Enter on Continue.



And that's it. If you wanted to create more RAIDs you could, but we can hit Finish for now.

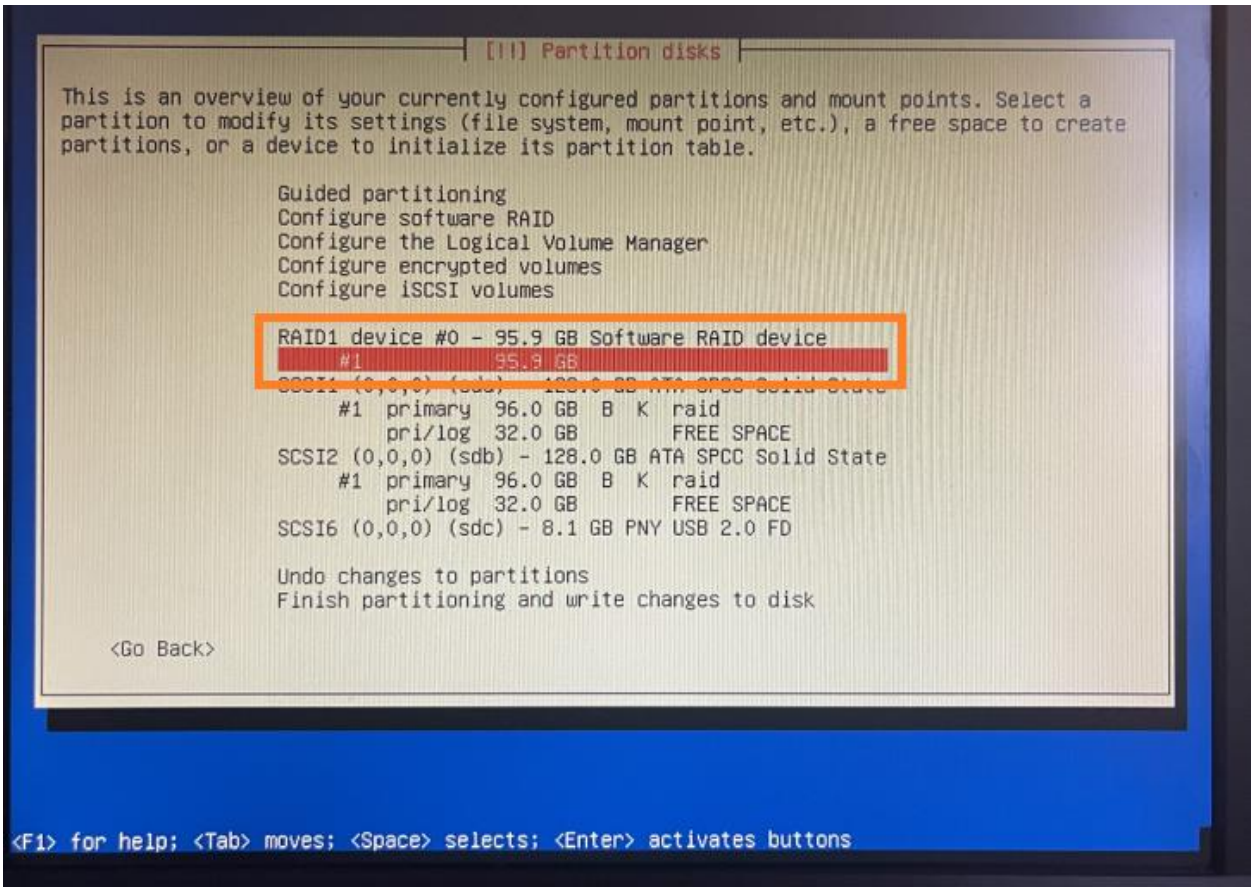


And we are now back to the main partitioner menu, with our spiffy new RAID up there and ready for us to do further things with.



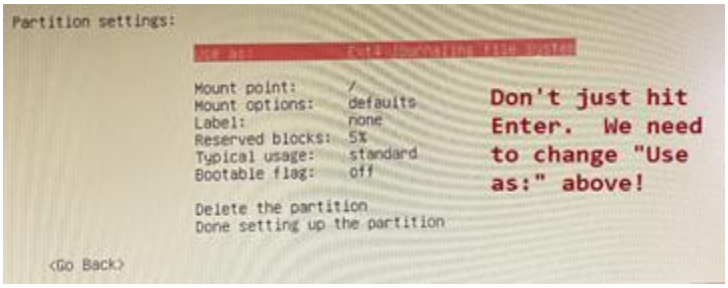
Step 3: Create an LVM VG containing 1 block device file: the RAID1 we created in step 2.

Now we're wanting to slap a VG on top of that fresh new RAID1, so we can fill it with LVs.

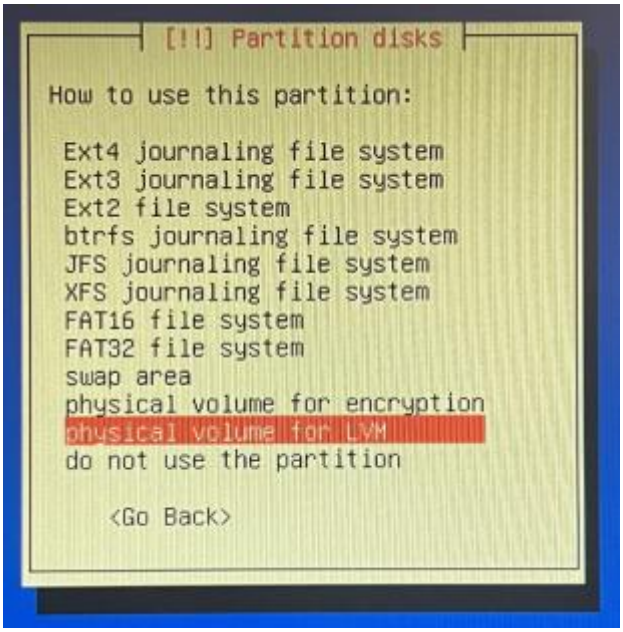


Press Enter here – we need to put a partition on the RAID-1 so that LVM will understand we want to put it in a VG.

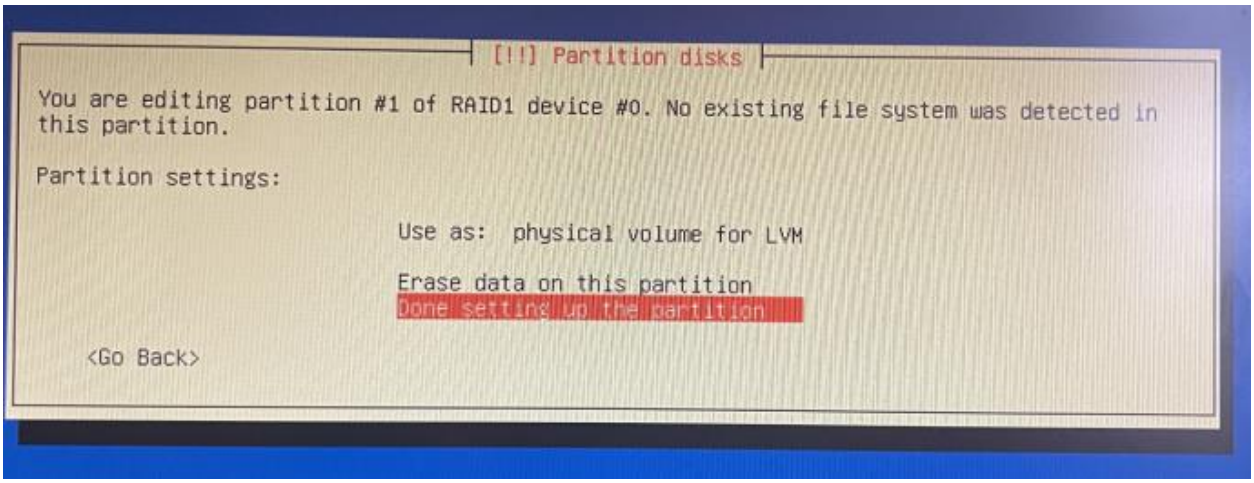
You'll see this screen again. Press Enter on "Use as:" …



Select “physical volume for LVM” here.

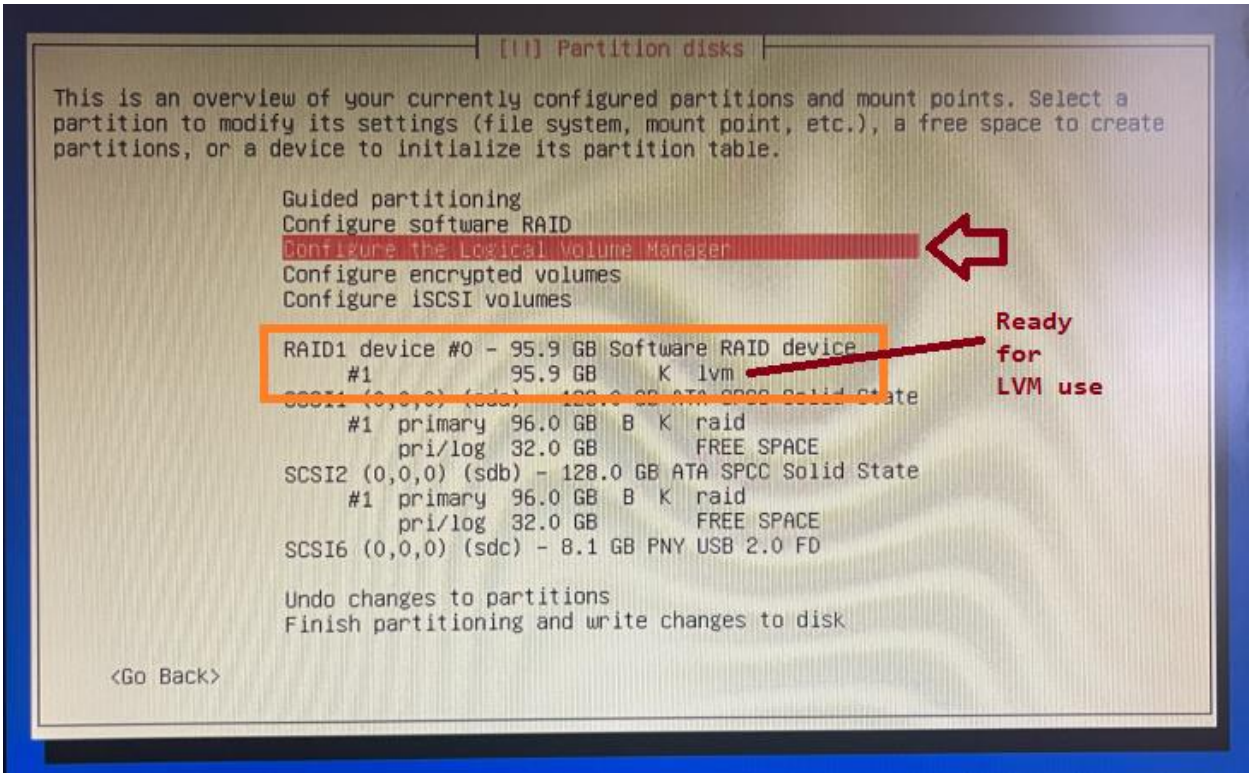


and then “Done setting up the partition”

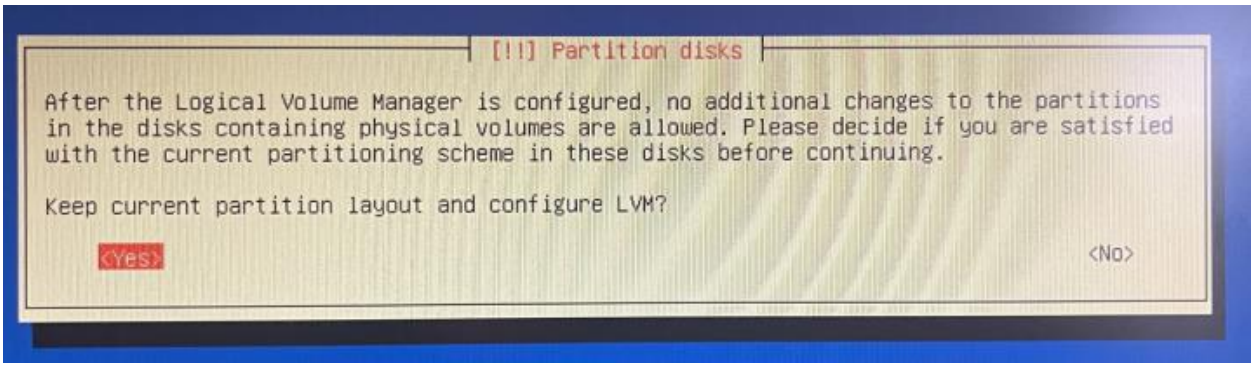


Now the RAID-1's partition type should be **lvm**. That doesn't mean it's an LV or anything quite yet, it's just ready to work with LVM.

So our next step is to highlight "Configure the Logical Volume Manager" and press Enter.

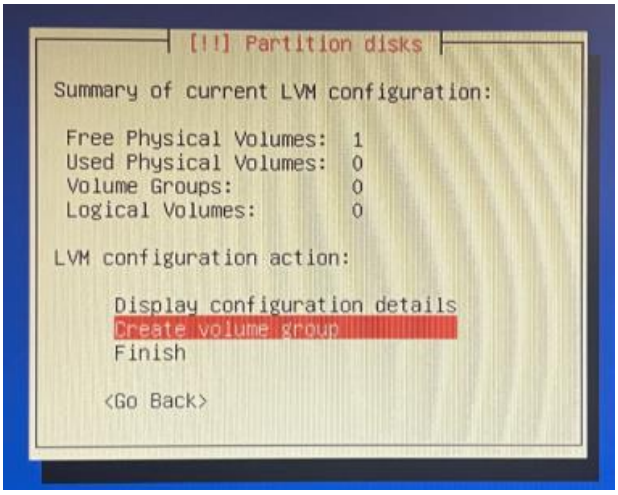


Another warning, similar to when we were setting up the RAID-1.

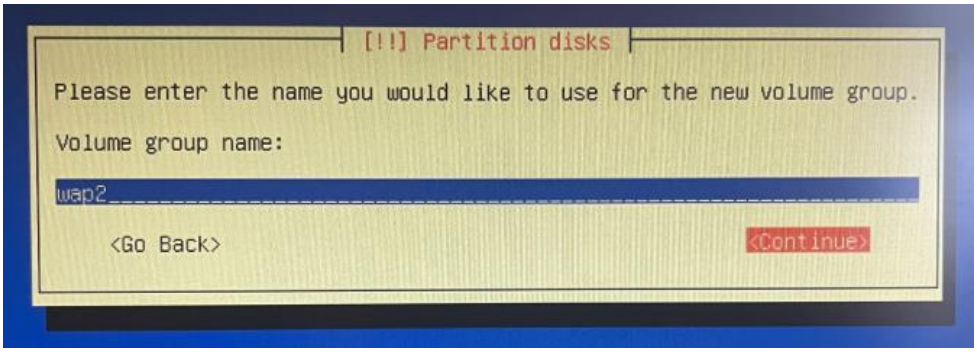


Here's the main LVM configuration menu.

If you guessed we want to "Create volume group", you are absolutely right.

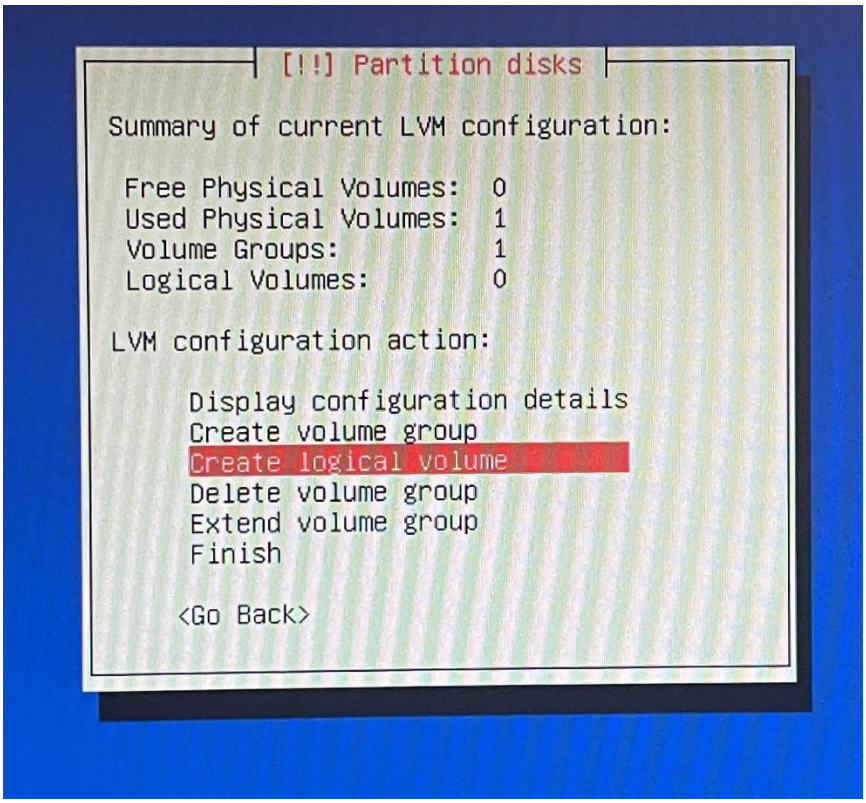


VGs need a name. In this type of setup where I'm just creating a single VG in order to use LVs, I usually give it the same name as the hostname.



Here, we can pick what block device files get to be in the VG. We just want to put the RAID-1 we made in there - `/dev/md0`. Don't put the FREE SPACE items in there. Then highlight Continue and press Enter.

And you'll be back in this menu, with some additional options because there is now one VG to do something to.



And now we're ready for the next step.

Step 4: Create several LVs in our VG for various Linux needs

Splitting up the Linux filesystem

It's definitely possible to install Linux using one partition or LV. The directory / will simply take up the whole one partition/LV.

Generally though, you want to at least set up a swap partition/LV (swap *files* are possible to setup but not within the Debian installer), so you're usually working with a minimum of two on the Debian installer.

Splitting it up provides 2 main benefits:

- It limits the problems full disk space can use in the rare event something goes haywire.
- You can also set specific mount options on the partition/LV. One of those options is `noexec` which prevents files from being set as executabe – this guide does not cover using that option but it is an example.

Splitting directories out into their own partitions/LVs does not otherwise really bring improvements, so it's far to consider this a bit of overengineering in this day and age.

- I've always liked to give `/boot` it's own small partition/LV (like 512mb) and set the "sync" option on it – making writes happen at the time requested instead of when Linux feels like it. This decreases the chances a sudden poweroff would make the system unbootable if it happens during an upgrade. This is likely overly paranoid thinking on my part in the SSD age.

Nonetheless, I like to separate out portions of the Linux filesystem at least like this.

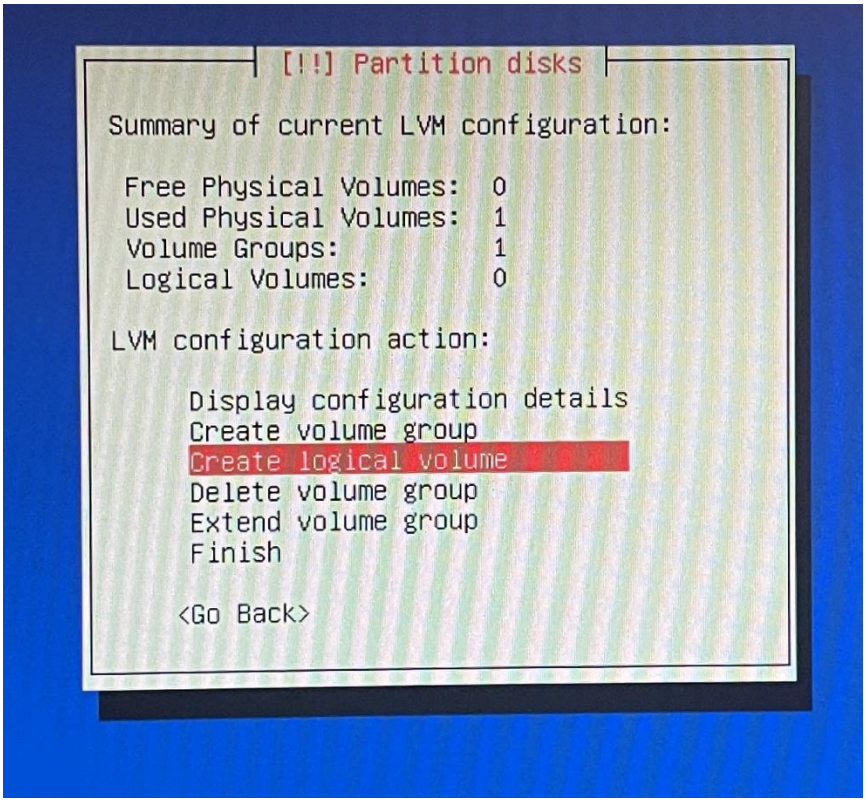
[swap]	1gb	If we set this up in Debian it has to be its own partition/LV
`/boot`	512mb	GRUB looks for Linux and other boot-related files here.
`/var`	8gb	"Variable data": Log files (`/var/log`), application data that isn't user-centered (like DHCP leases, caches, apt updates, etc.) and temporary stuff that needs to survive reboot (`/var/lib`)
`/tmp`	2gb	Temporary stuff, does not need to survive reboot
`/home`	1gb	User directories
`/`	Rest of space in VG	Everything else, including all the important Linux system directories like `/sbin`, `/bin`, `/lib`, `/etc`

Since this is my guide and we're doing things my way, we'll be making 6 LVs in the VG we made in the previous step.

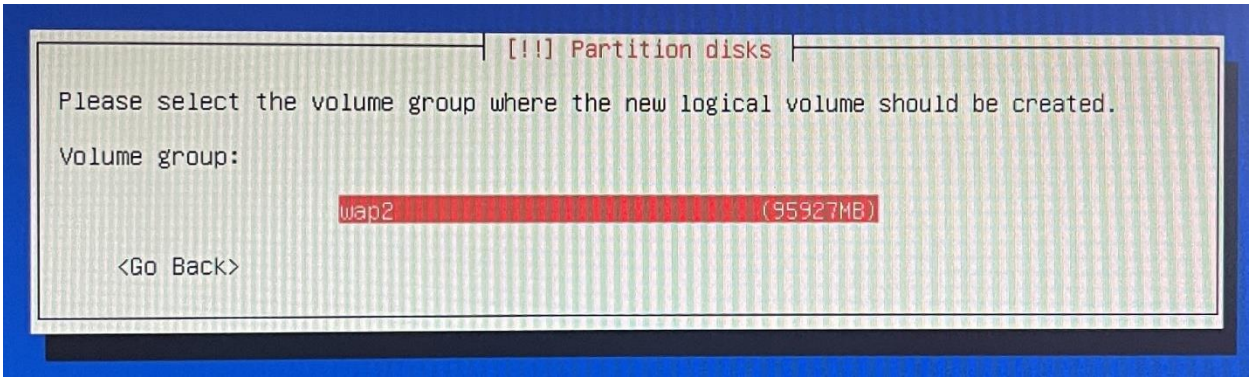
Making the /boot LV

We're going to be creating the LV for **/boot** first.

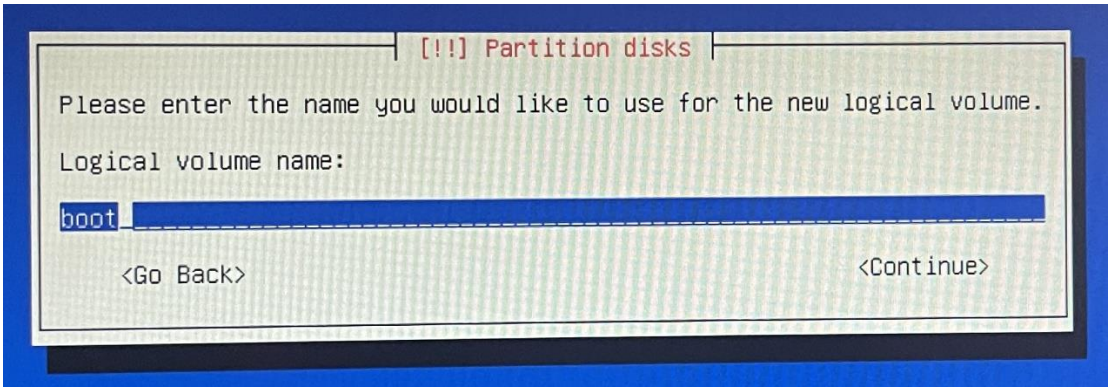
So you should still be here. Press Enter.



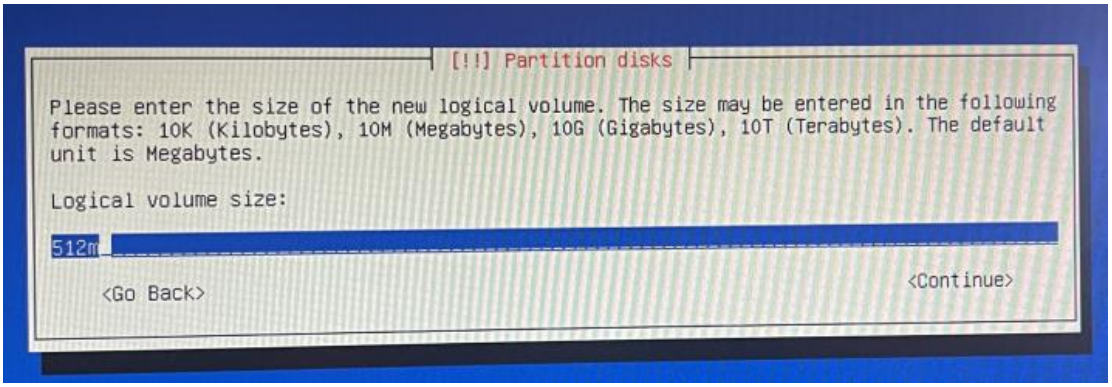
You'll be asked which VG you want to work with. We only have one so this is an easy decision.



LV’s require a name. I like to name it the same as the directory it’s for.



Then you’ll enter the size of the LV.

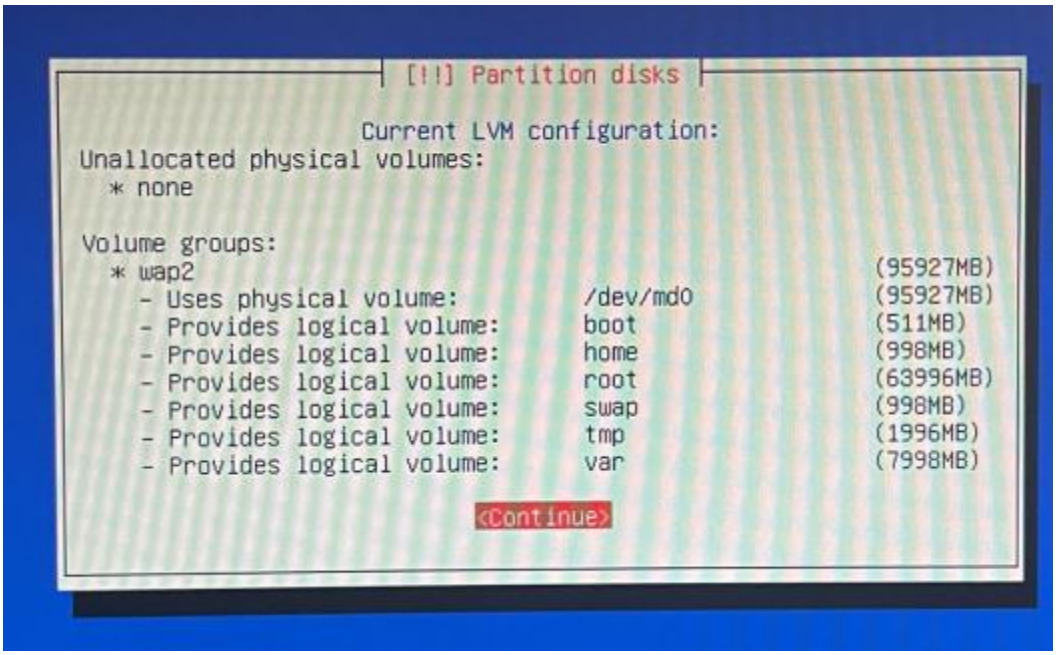


And that’s it, you’ll be back at the LVM configuration screen.

The other LVs

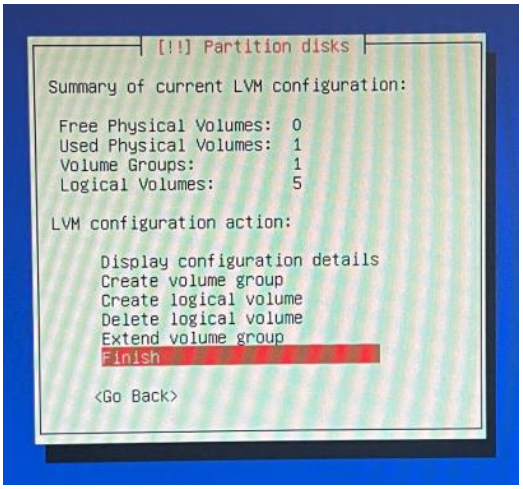
Repeat the above for the other LVs, using the size in the table shown earlier.

When all are setup, you can select “Display configuration details” to verify that all the LVs are correct. If something is not right, delete any LVs that are wrong and recreate them.



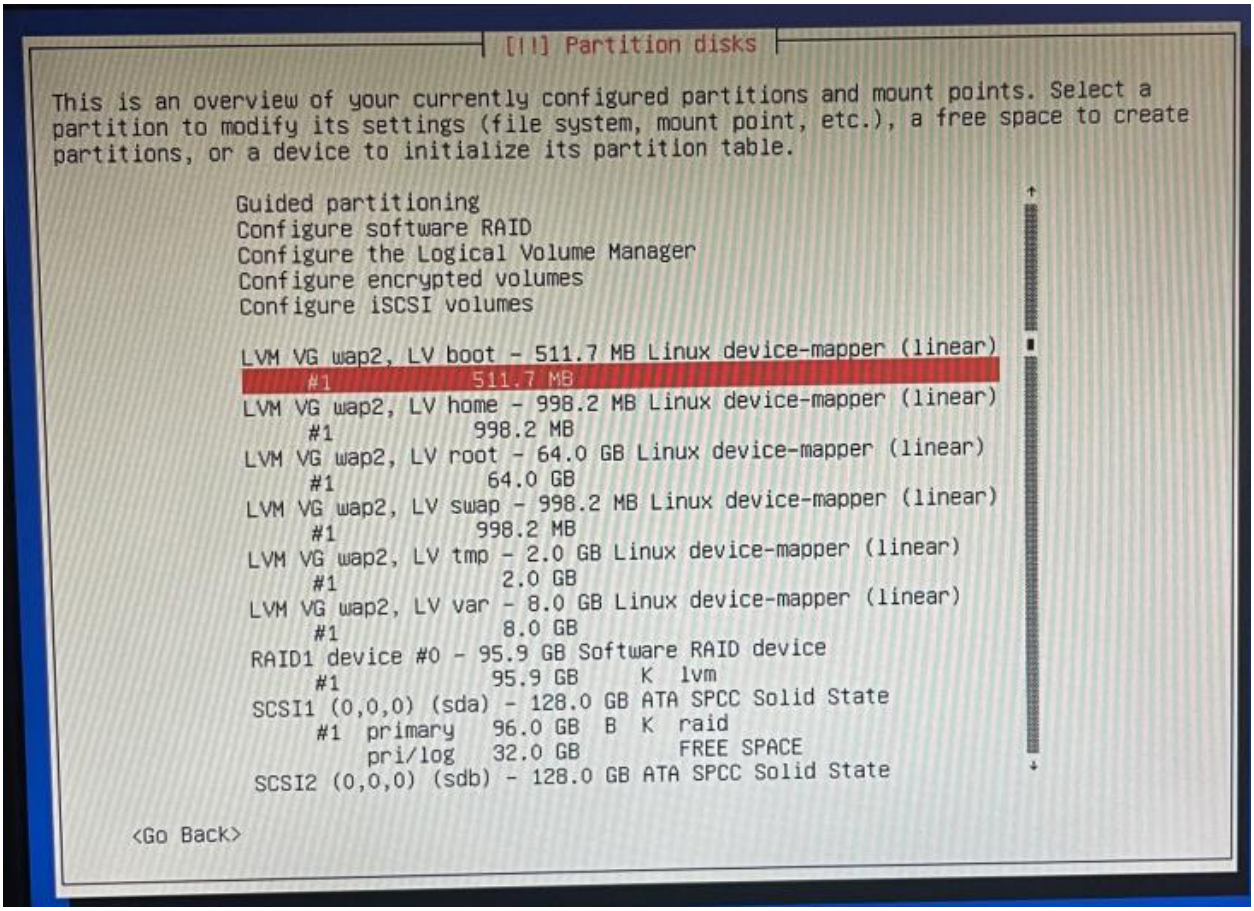
Then we hit Finish here.

(The Logical Volumes says 5 here because it’s an earlier screenshot and I forgot to do one of the LVs. It should say 6.)



After hitting Finish, you're back to the "Partition disks" screen.

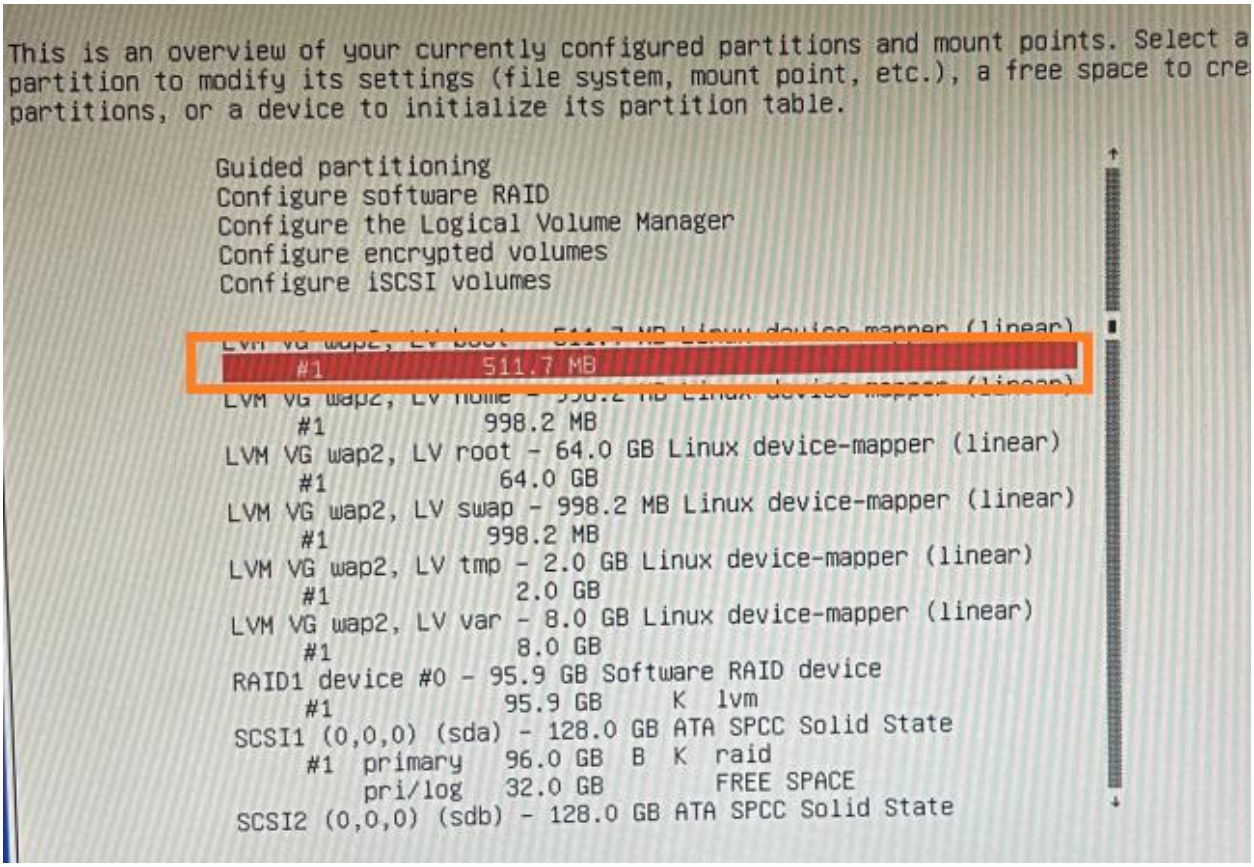
All of the LVs have joined the partition party now, and we can now assign mount points/things to do to them.



Step 5: Assign mount points/purposes to each LV (except swap)

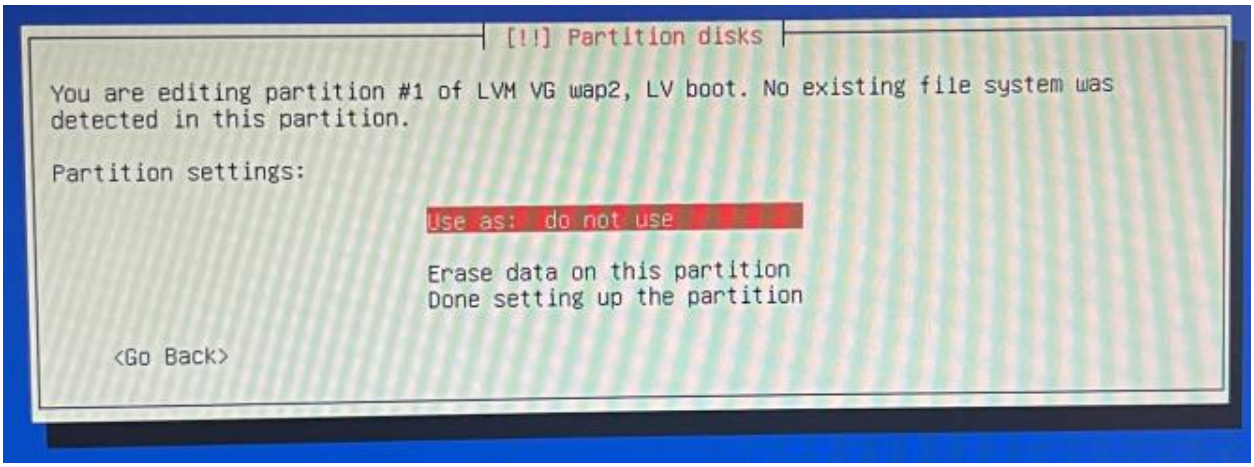
We need to do this for each LV – for swap we'll do something special. We'll start with the one for `/boot`.

Move the highlight here and press Enter…

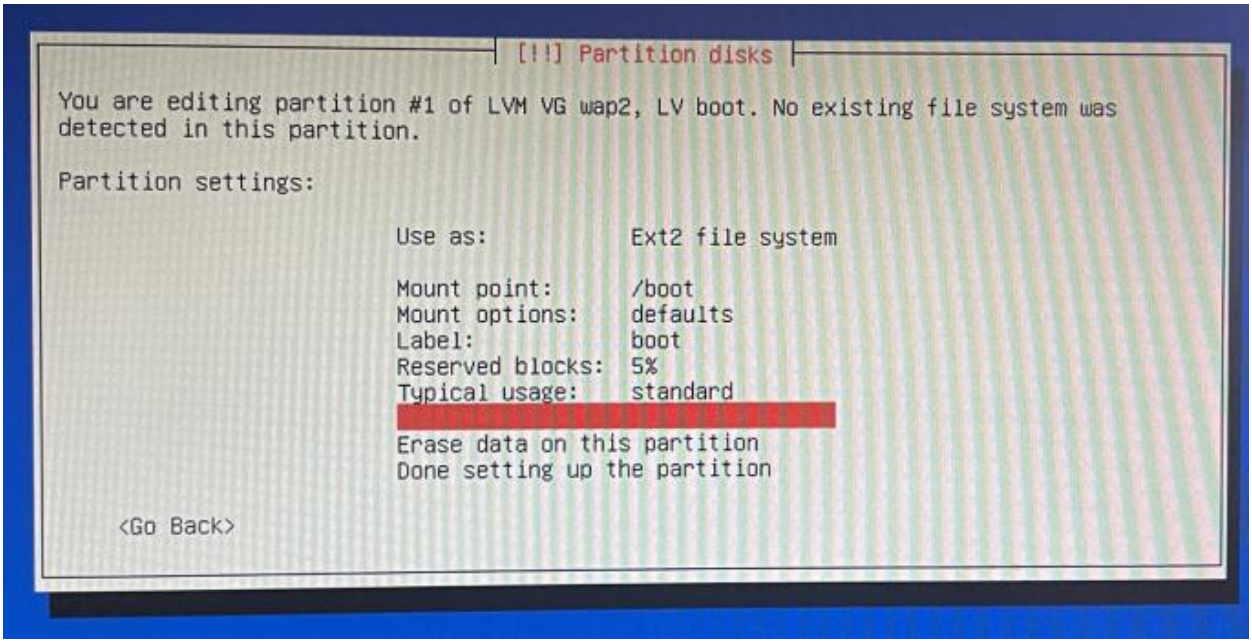


You're now editing that object.

Press Enter on "Use as:" to change it.



Change the options for /boot to look like this:



"Use as:" filesystem

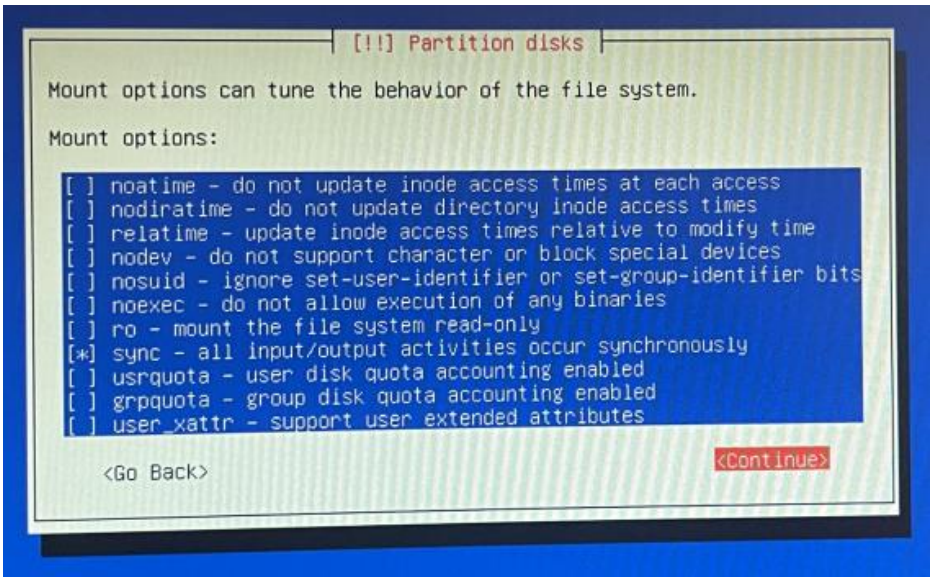
I have a habit of using ext2 **just for `/boot`** but not sure it's needed for modern GRUB (it probably can use Ext4). The other partitions should use ext4 at least.

Label

I like to make the label match the partition name.

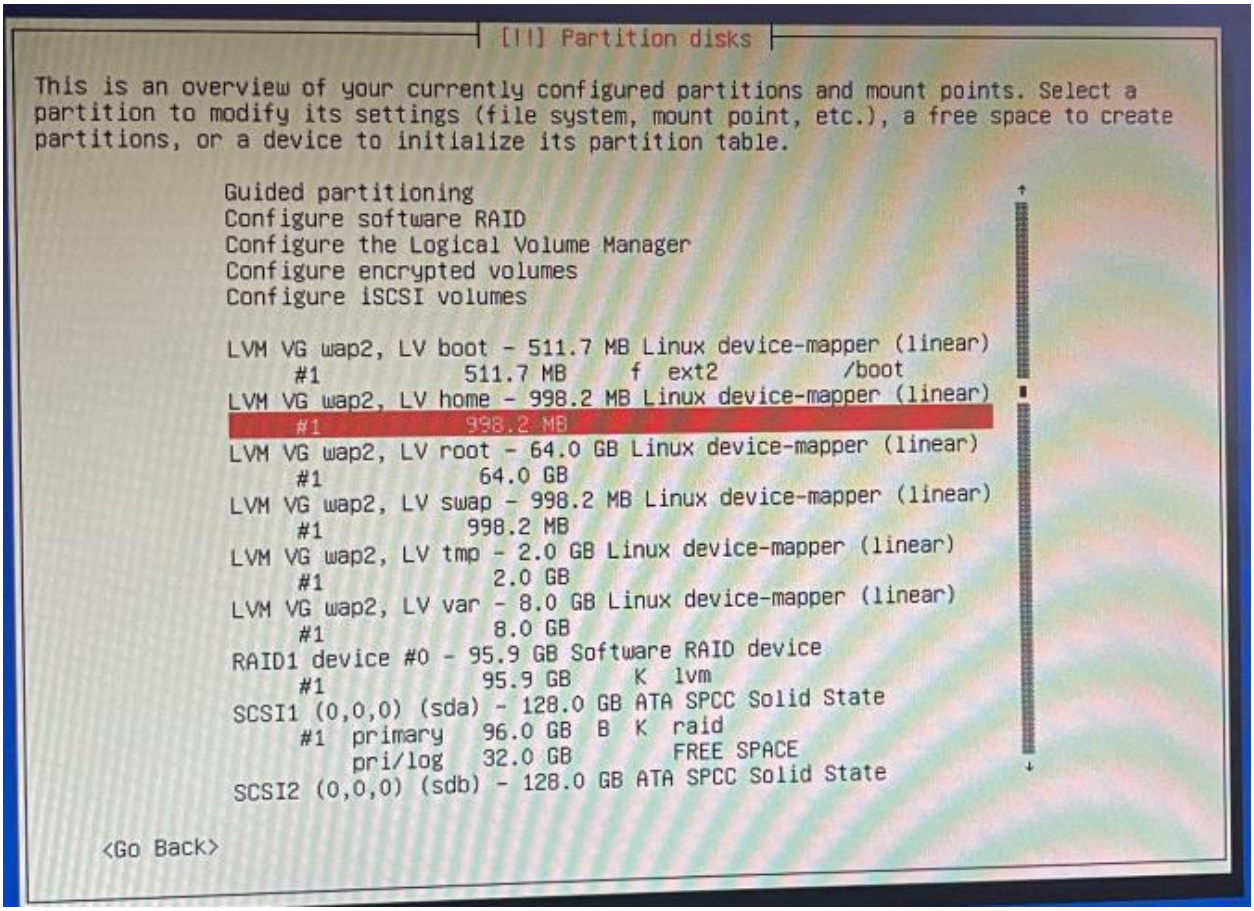
Mount options for /boot

For `/boot`, I like to enable the "sync" option, but not the other options.



Then move the highlight to "Done setting up the partition" and press Enter.

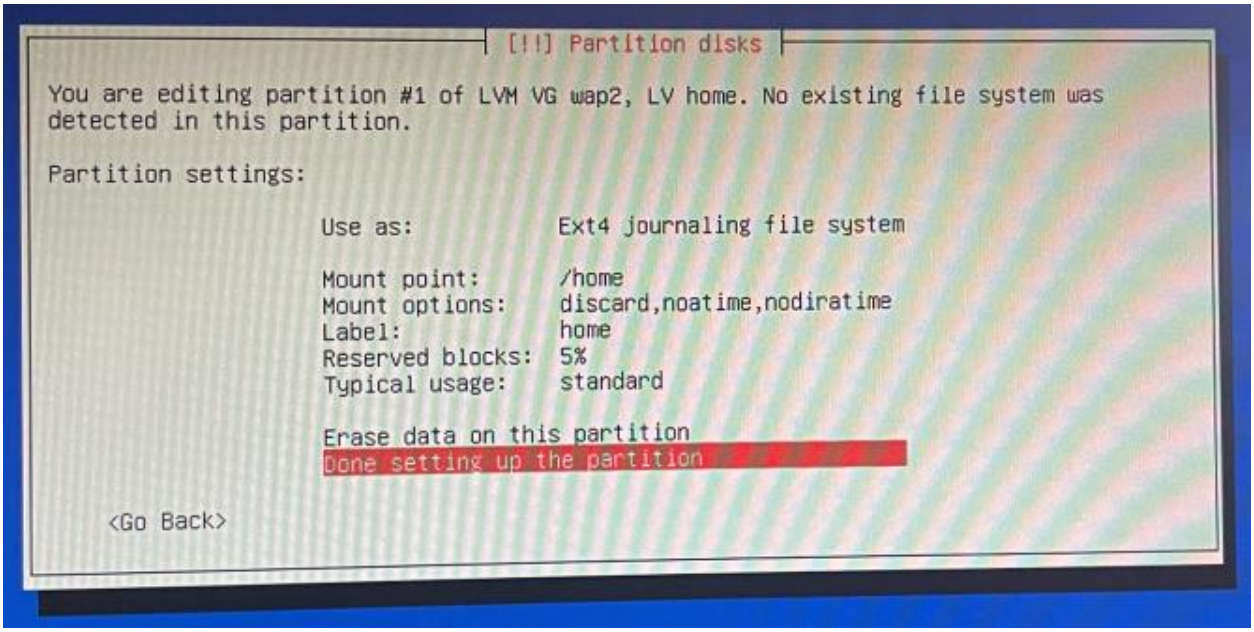
And now we can see `/boot` is setup.



Other directories except swap (`/var`, `/tmp`, `/home`, and `/`)

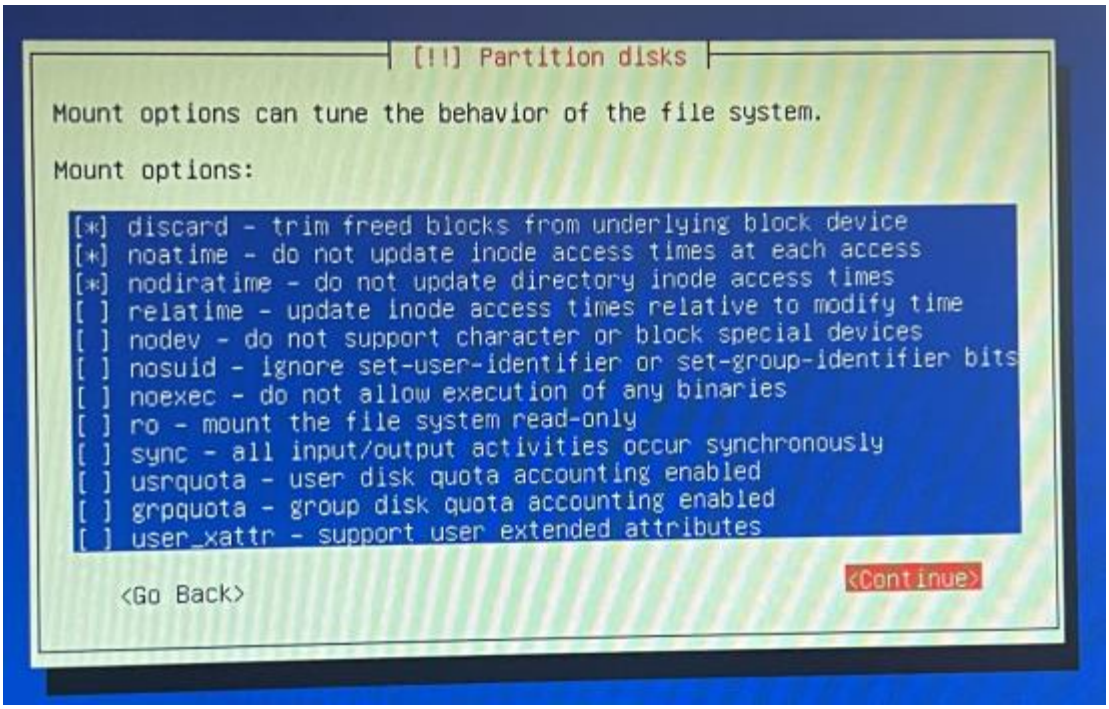
Basically we need to repeat the above for the directories `/var`, `/tmp`, `/home`, and `/`.

We want to use at least ext4 for the other LVs. `/home` is presented below for illustration.

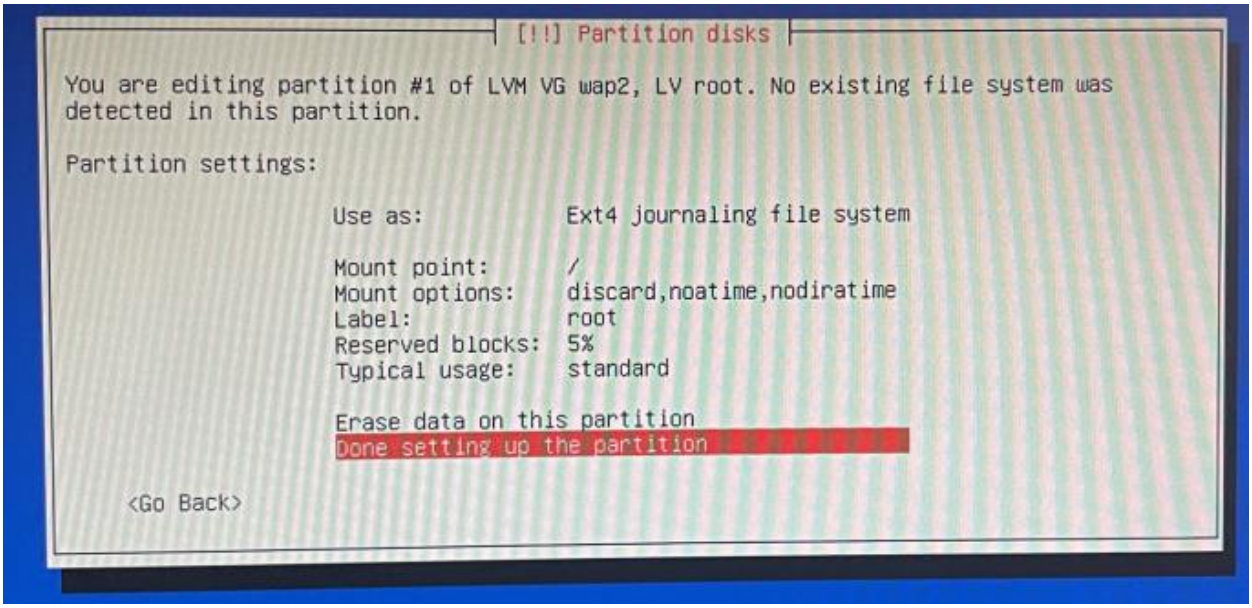


And for the other LVs, these are the mount options I use.

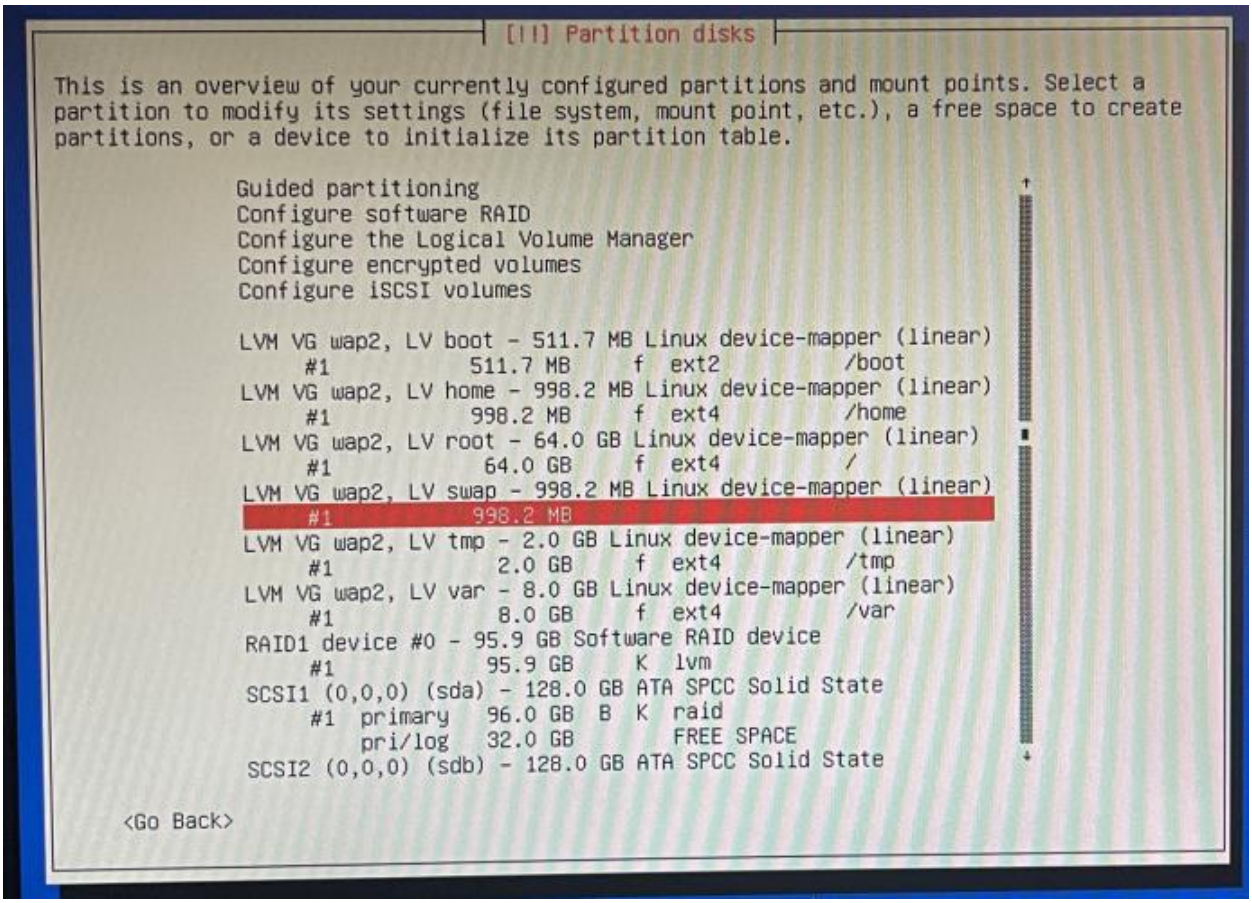
- `discard` enables TRIM (honestly should have done that for `/boot` too … it can be fixed later),
- and the `noatime` and `nodiratime` options will improve performance and increase SSD endurance by skipping updating access times any time a file is touched.



For the directory /, I give it the label “root.”



When all except the swap LV are done, you’ll get something like this:

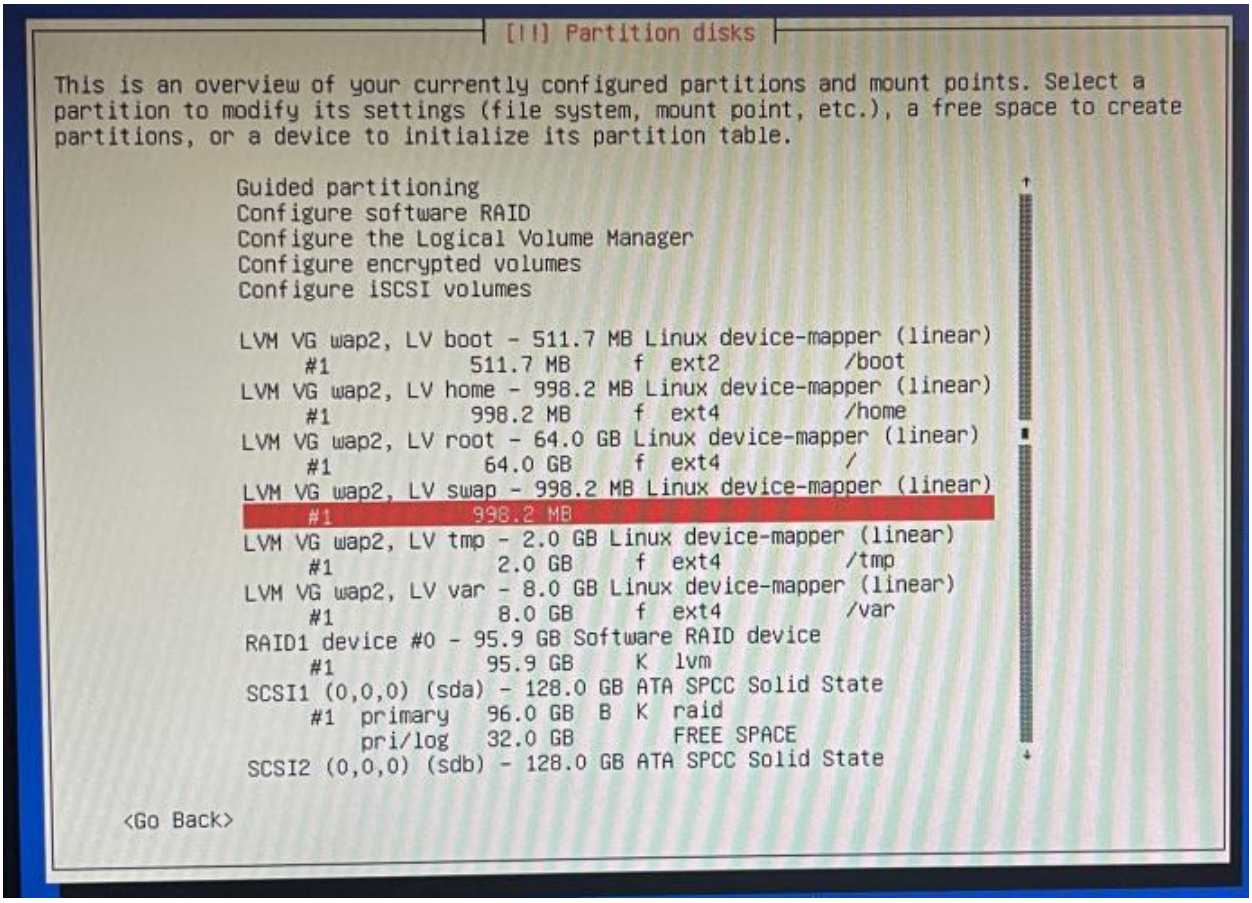


Ready for the next step.

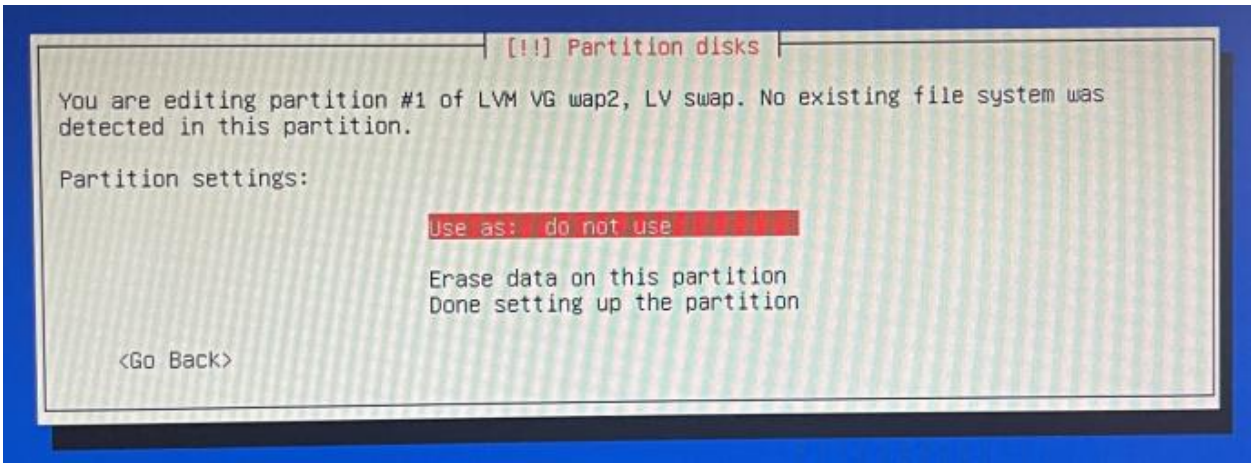
Step 6: Encrypted swap setup

What we're doing here is setting up an encrypted swap file – the encryption key is unique and generated during each boot. This means across reboots, whatever was in swap is effectively unreadable, because again, that key is not saved anywhere. So any applications that might have had private data that got put in swap – that information will be protected.

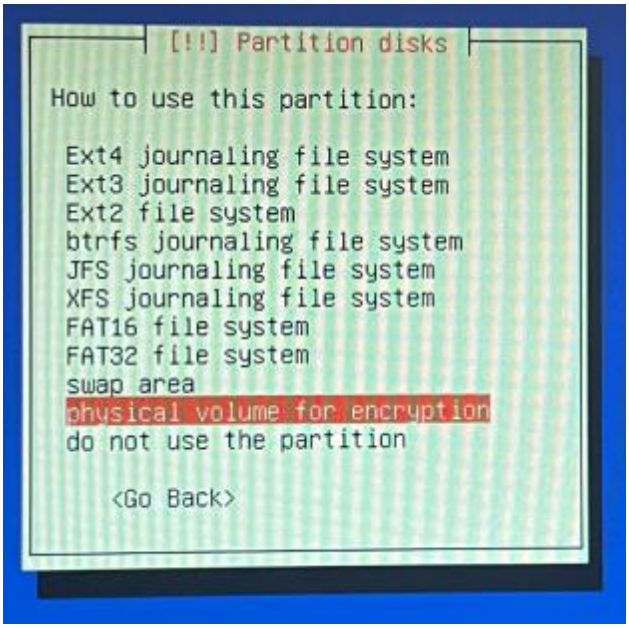
From here …



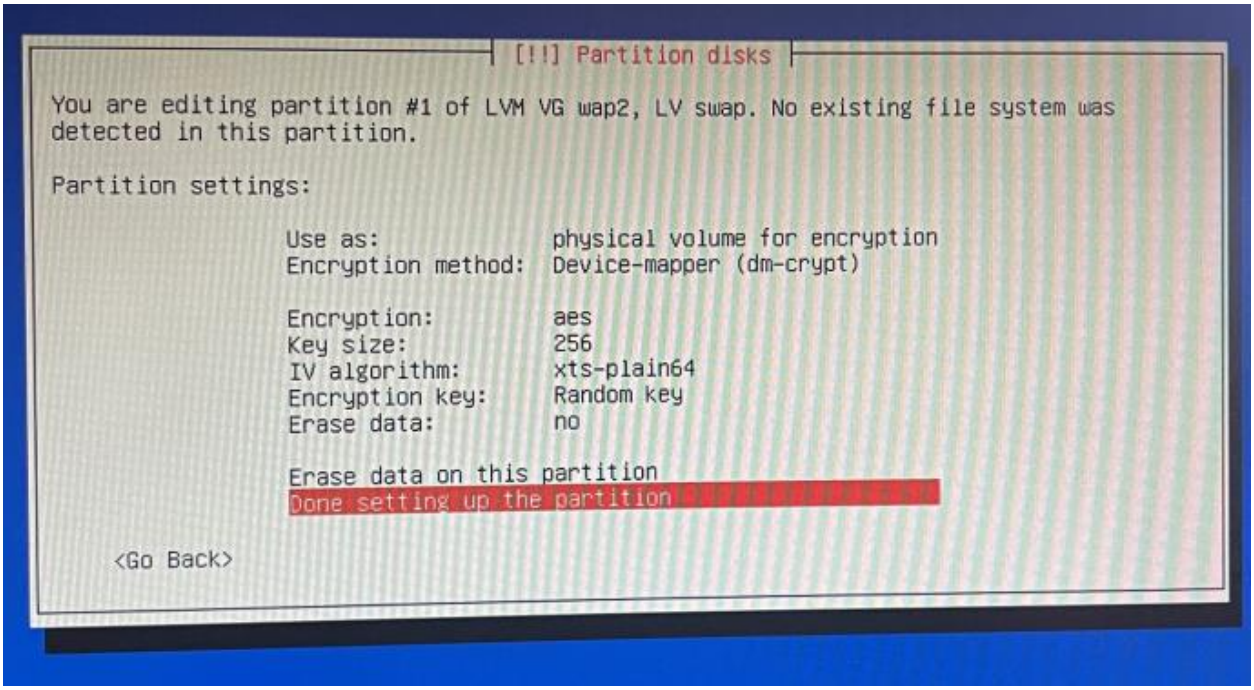
press Enter on that unallocated swap partition, and you'll see this:



We'll change the "Use as:" to "physical volume for encryption."

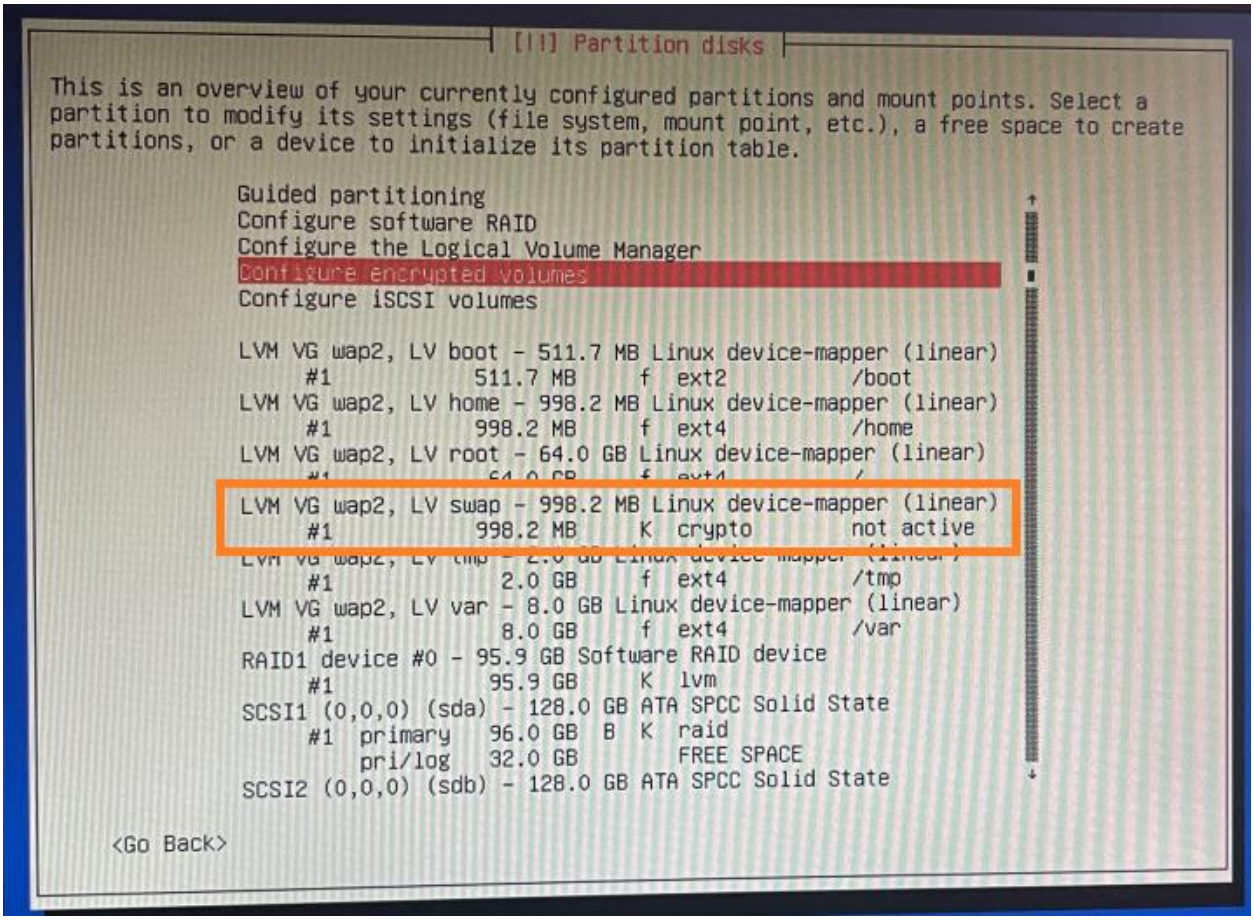


You'll see a popup about loading some components for encryption, and then this:

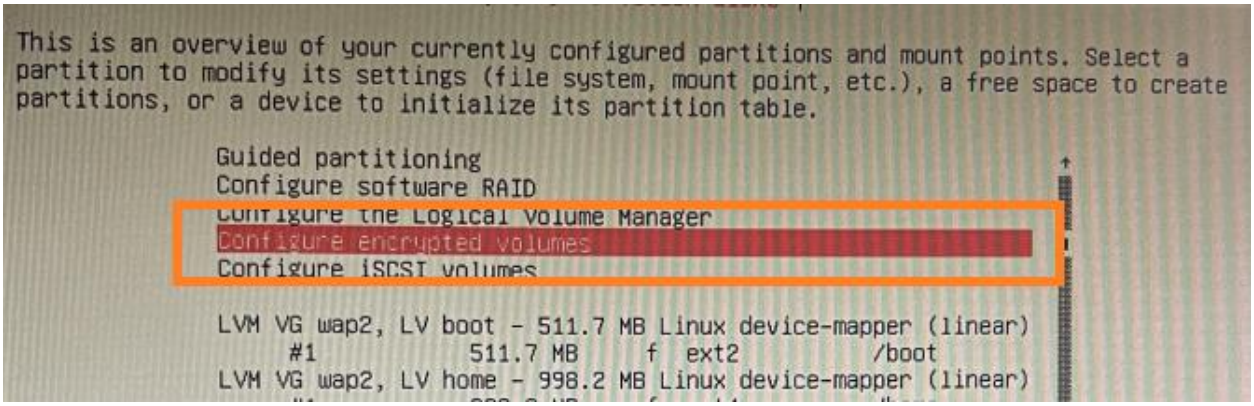


Of this, just make sure "Encryption key" is "Random key", then say you are "Done setting up the partition."

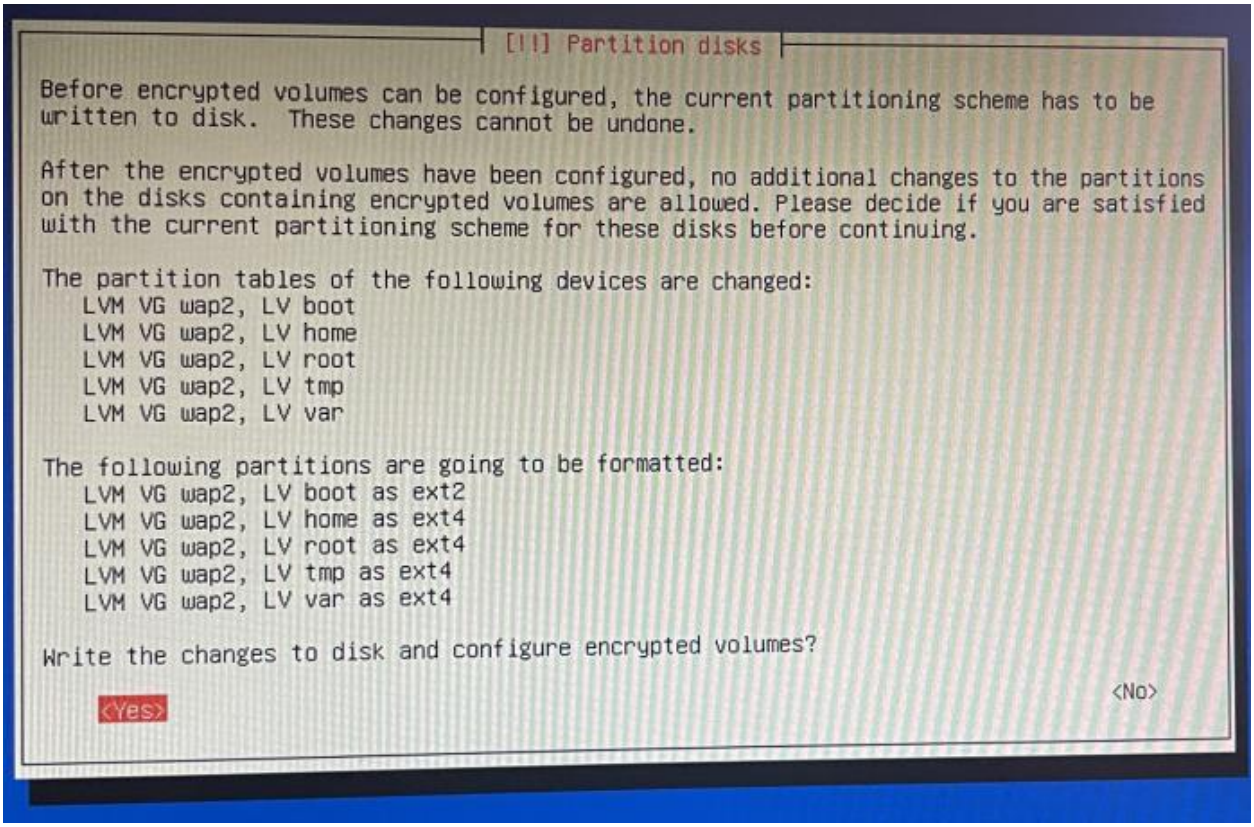
All right, now it says “crypto” and “not active.” Time to make it active.



Now highlight “Configure encrypted volumes” and press Enter.

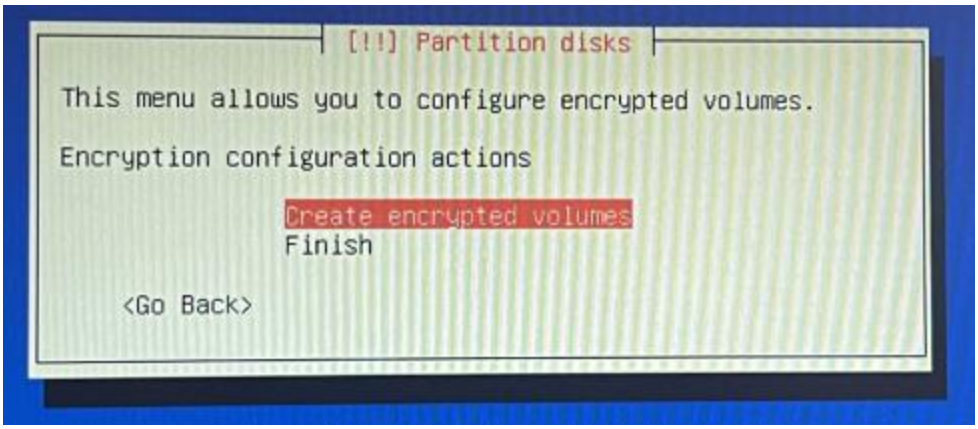


This will be the last time you see this warning, if you are following this guide.

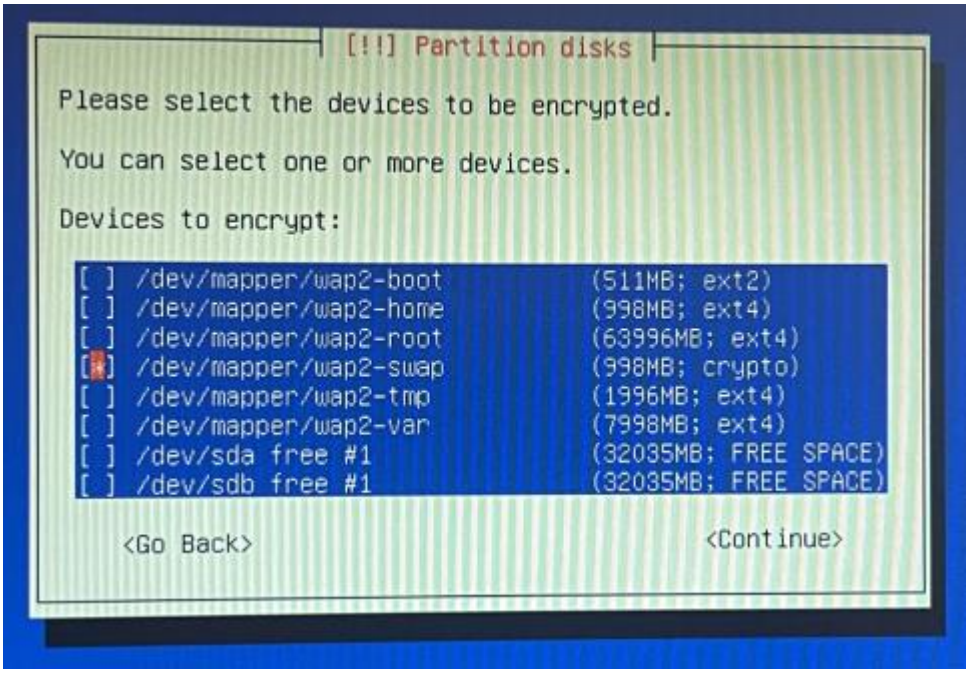


Say Yes here.

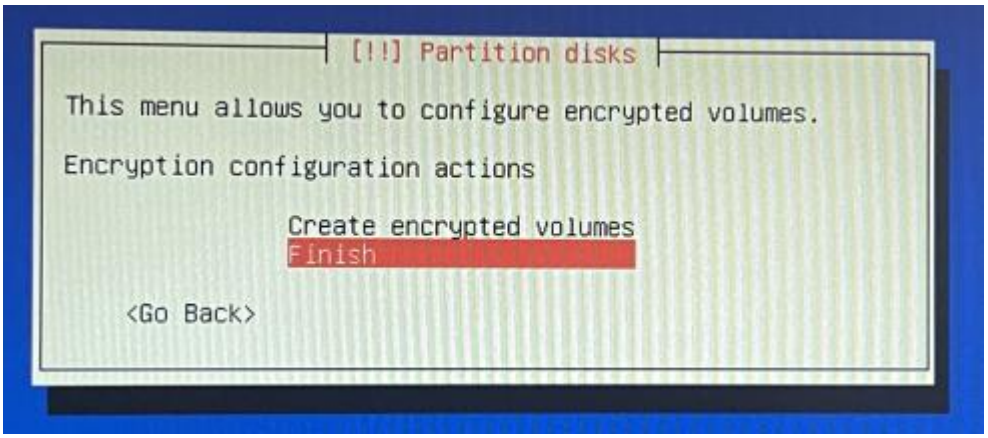
Then say “Create encrypted volume” here.



Check the one that says “crypto”, and none of the other ones.

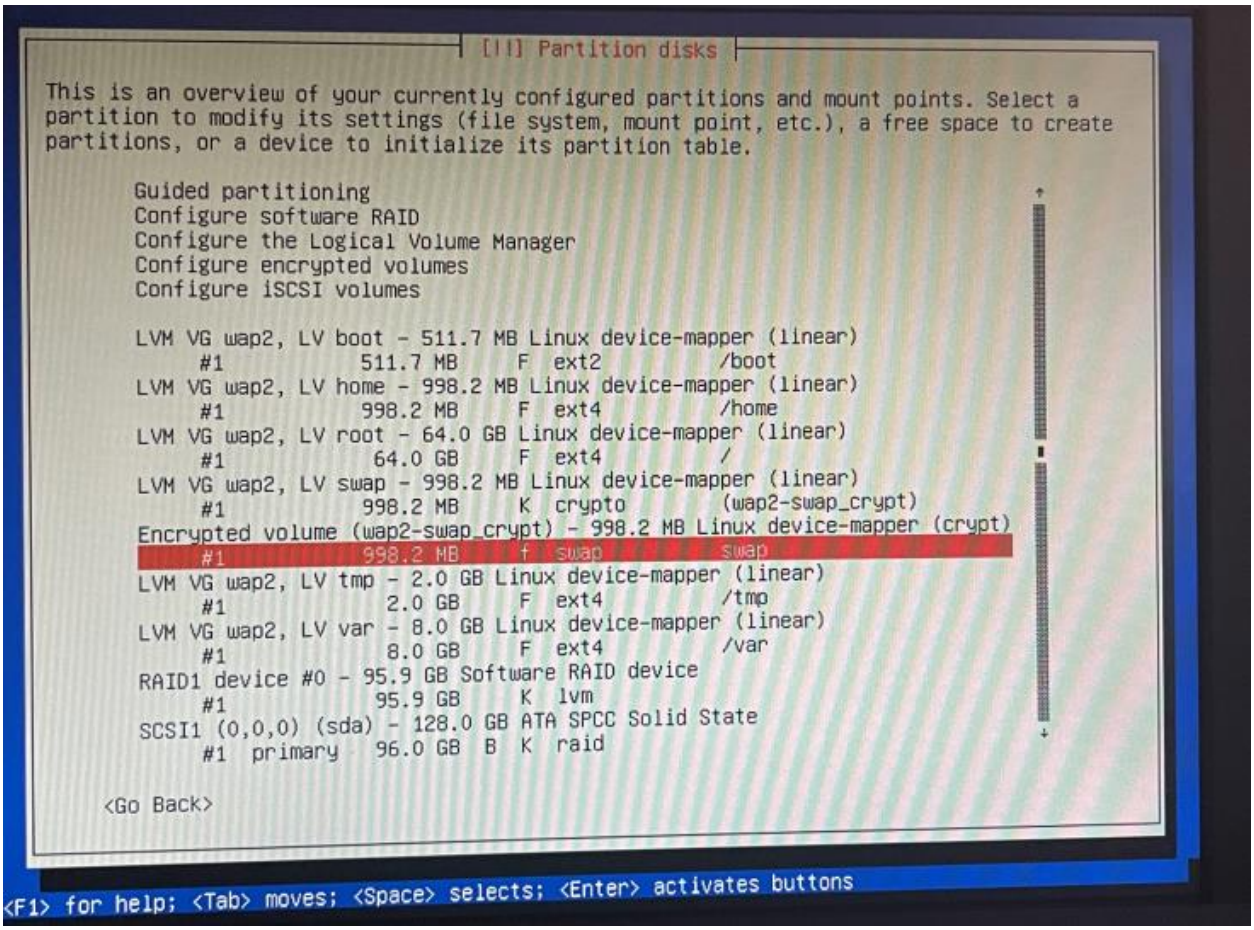


and then hit “Continue.”

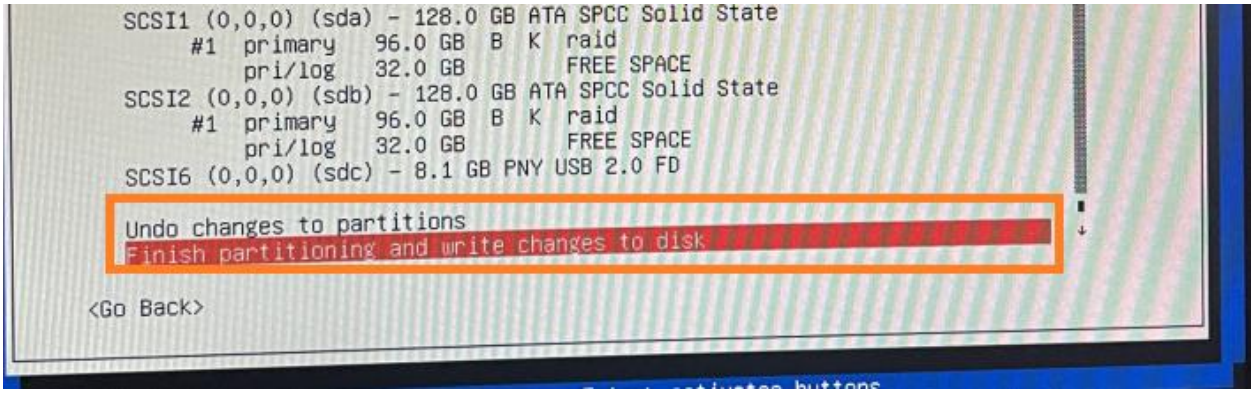


Then “Finish.”

I do believe Debian will assume (at least the first) random-key crypto partition you make will be a swap partition, so it sets it for you.

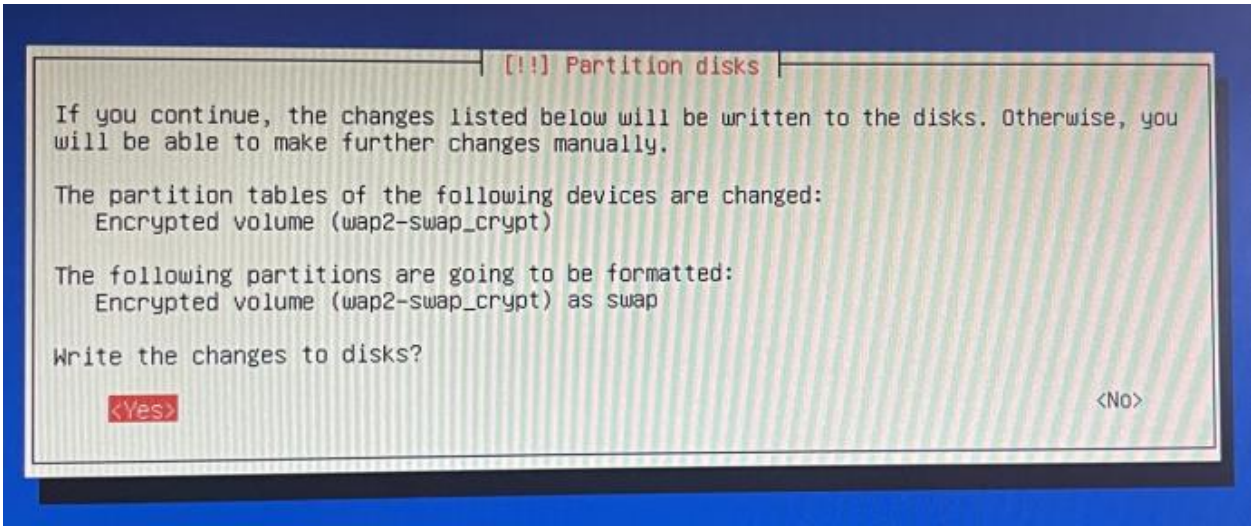


Scroll all the way down and select “Finish partitioning and write changes to disk.”



You aren’t done with the install yet, but it’s easy from this point on.

Of course, *one* more warning.



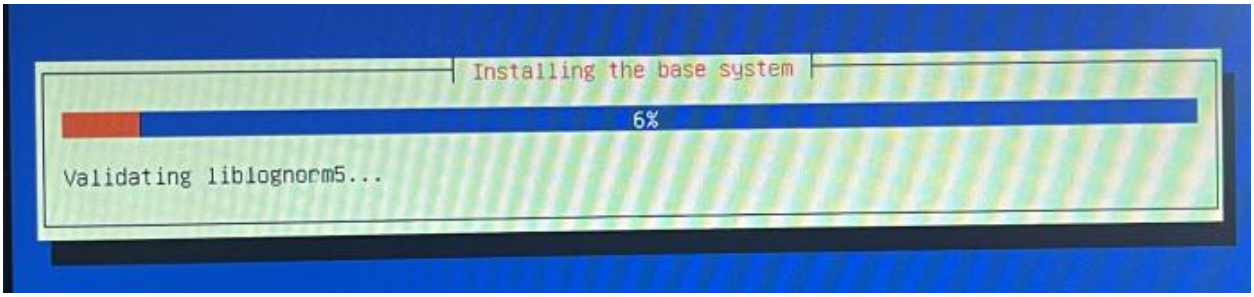
Say “Yes” here.

Installation Actually Begins

Installing the base system

Now that we've defined a home for everything, the installer will actually start doing some installation.

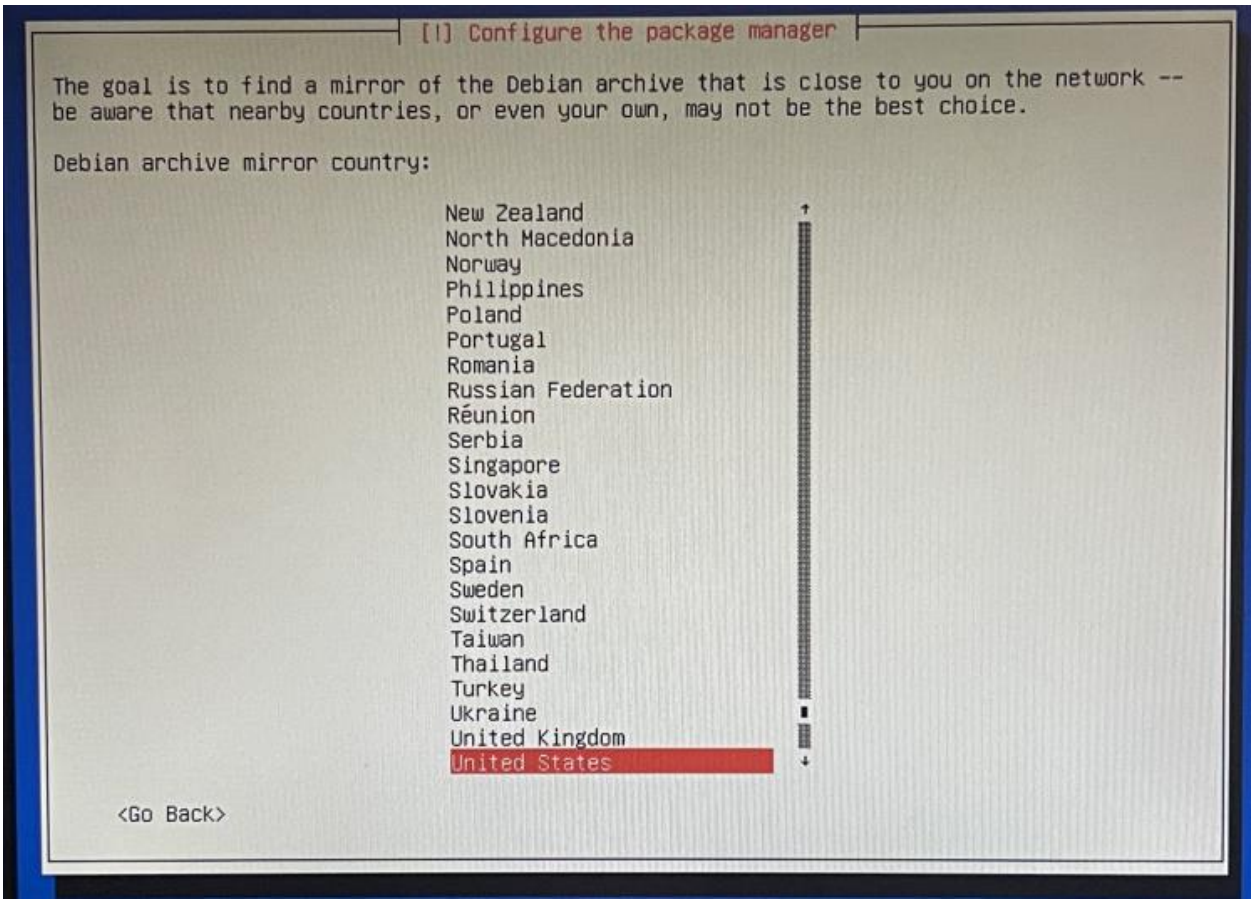
This will progress for a while.



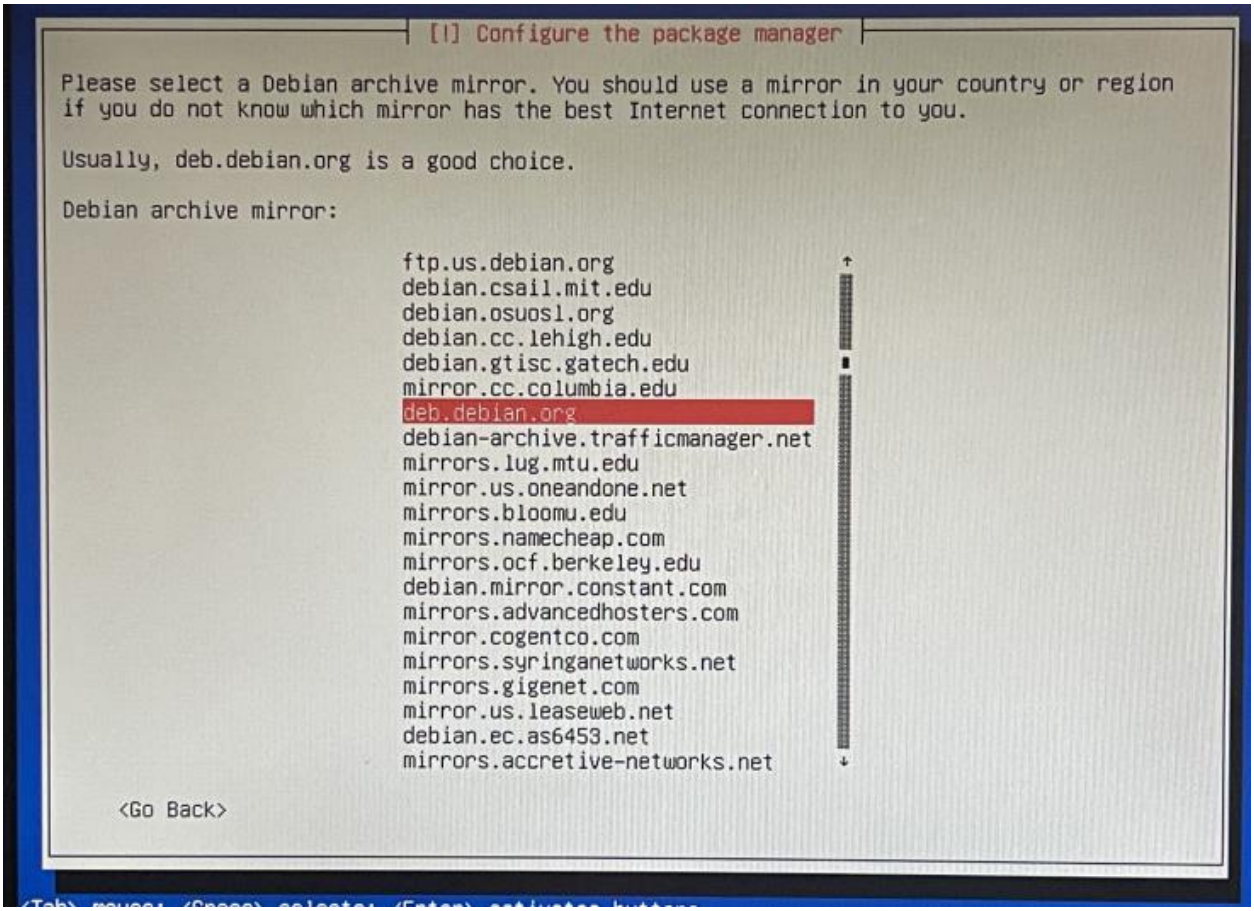
Package manager configuration

Now the installer wants to know which mirror we want to use to download updated and any extra packages from that weren't included in the netinst image.

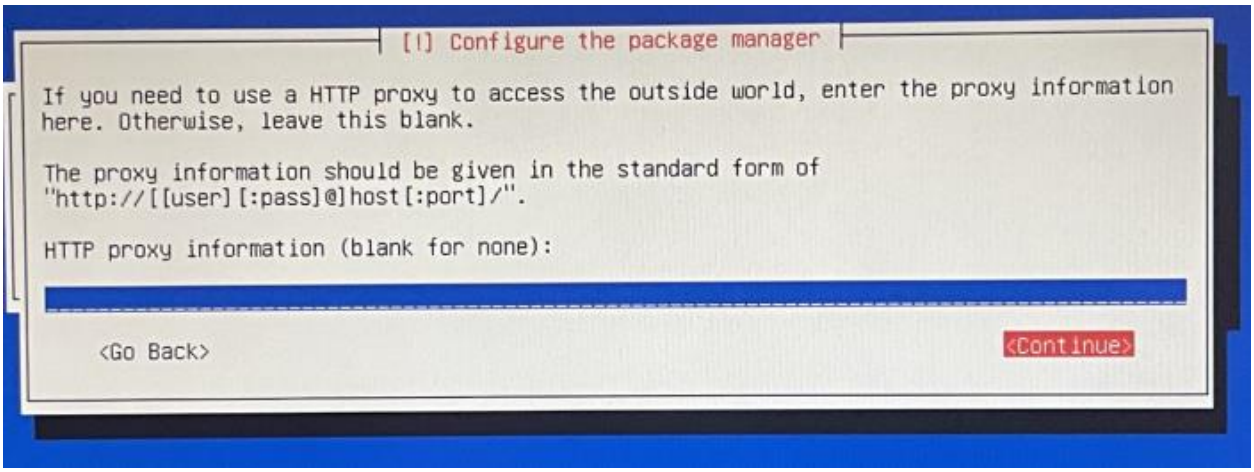
Press Enter here (or select your country and then press Enter).



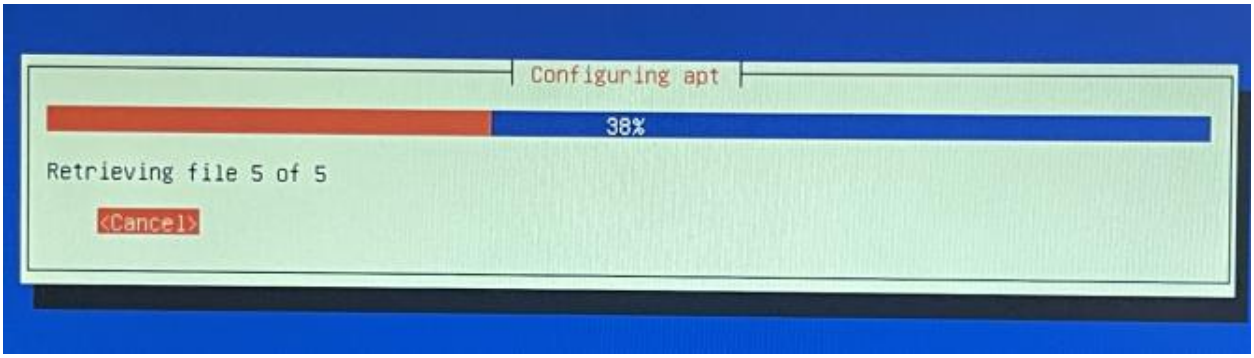
I just take the installer's suggestion here and press Enter.



Say "Continue" here unless you know you need an HTTP proxy. If you are a typical residential Internet user, you do not.

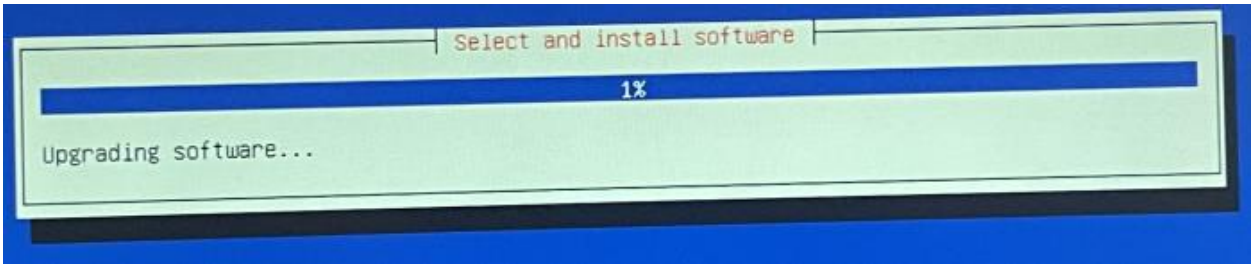


It’s getting package lists from the mirror now.



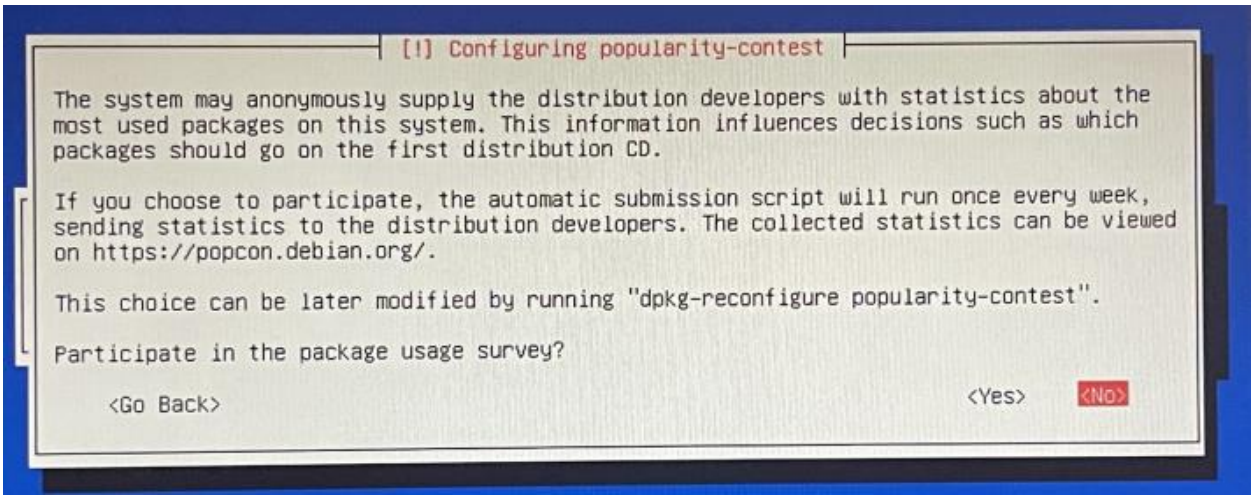
Applying latest updates of stuff on `netinst`

Next it will check for and install newer versions of anything that was included in the `netinst`…



Popularity Contest Question

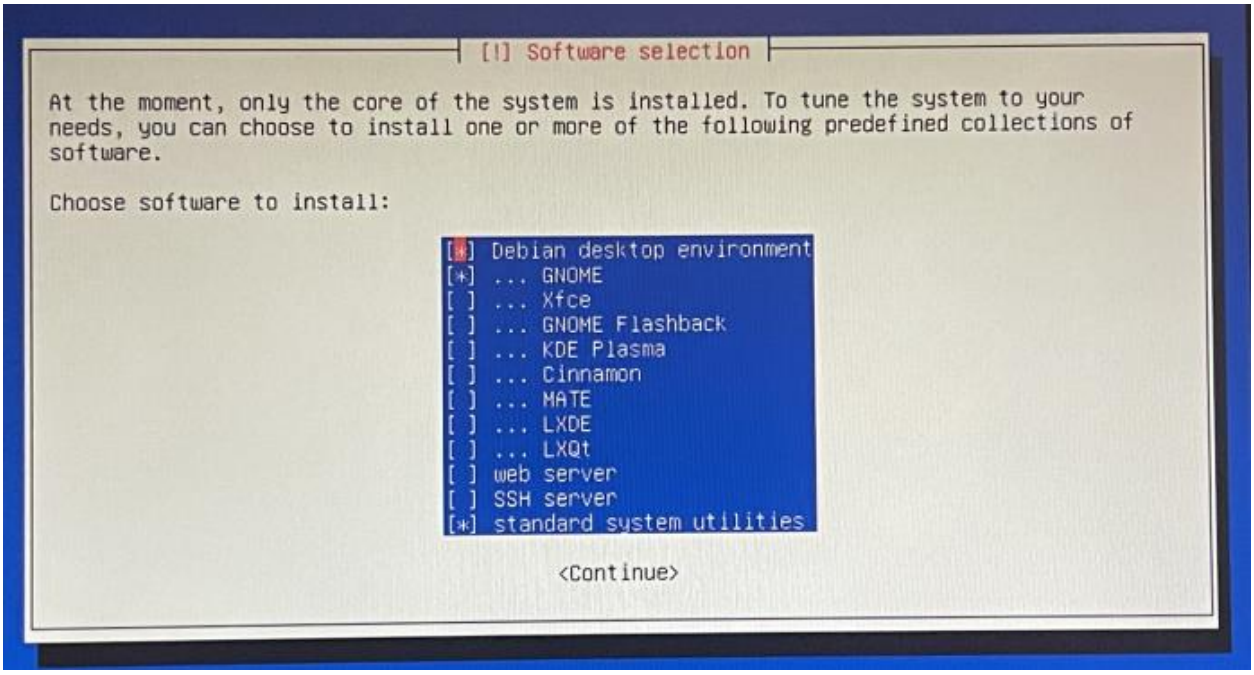
I normally wouldn’t mind helping Debian out, but this is an appliance-type setup, so I’ll say No here.



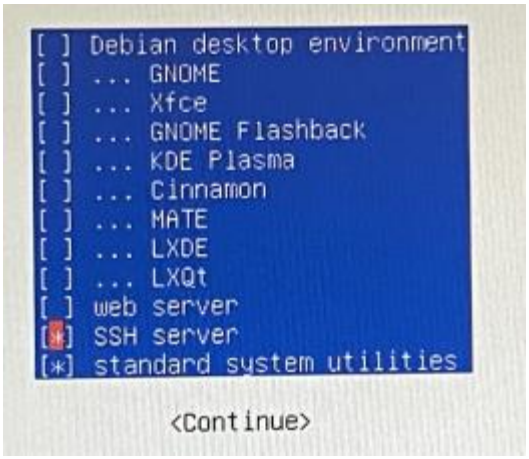
Software collections

Here you are presented with a list of “software collections.”

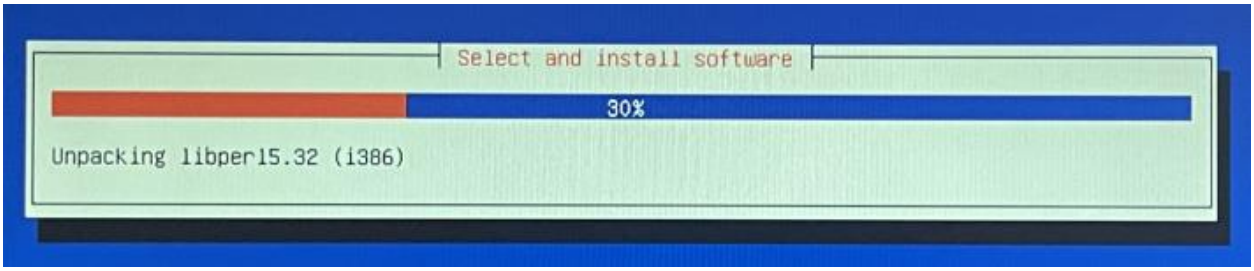
If you wanted to install a desktop environment, you could check one. There’s also a few other useful collections.



For appliance use, we won’t be needing any desktop environment. All we need is “SSH server” and “standard system utilities” on there.



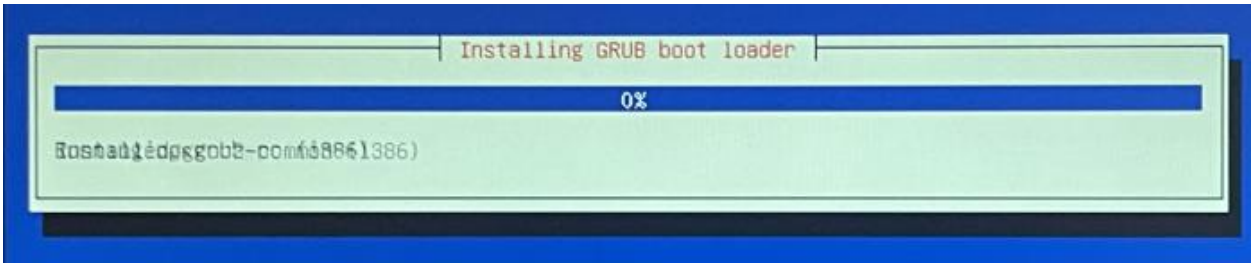
Make sure just those two are checked, then “Continue.”



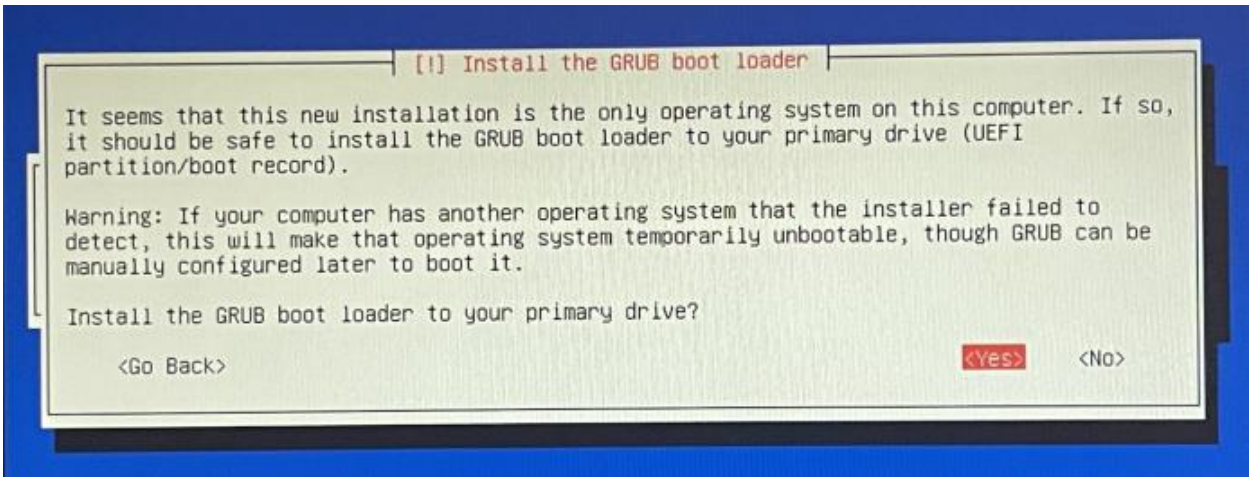
Noticing how that says i386? It's because I goofed and accidentally used a 32-bit version of the netinst .iso I had laying around. 64-bit should say amd64. Derp.

GRUB bootloader install

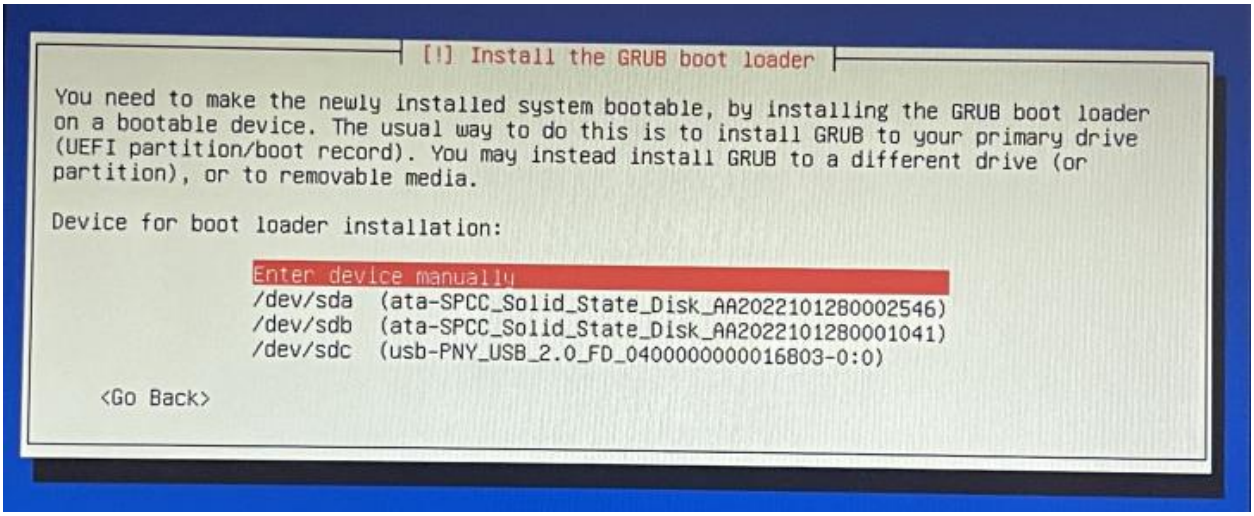
When it says it's installing GRUB, you're almost to the end.



If no other OS is detectable. you'll see this. Since these are virgin SSDs, there won't be. You want to say "Yes" here.

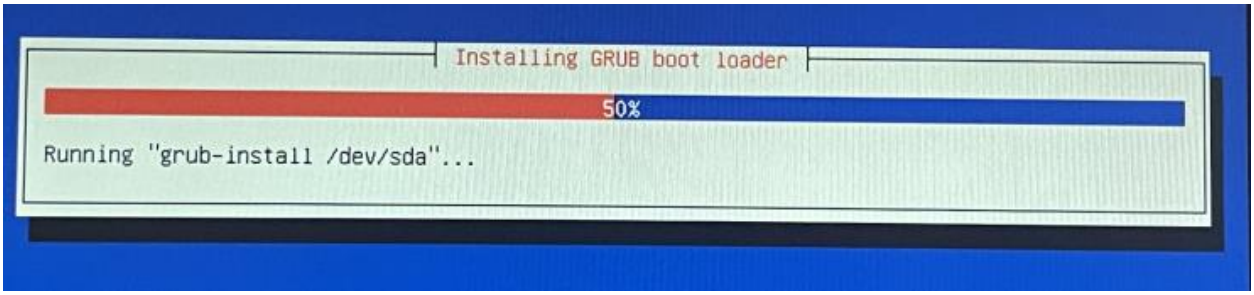


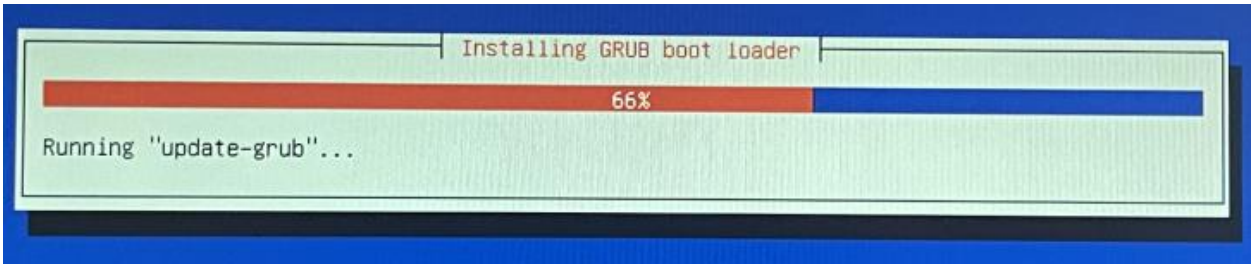
Now the installer asks to which physical disk we want to install the bootloader on.



We really want to install to both (so that the system will boot off of the other drive if the first stops working), but we'll pick `/dev/sda` for now, and do `/dev/sdb` after installation.

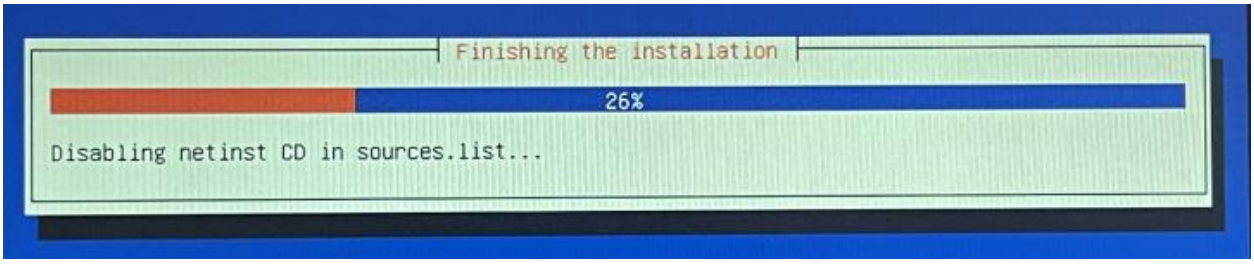
It'll take a moment to install.





Installation Home Stretch

You'll see this.

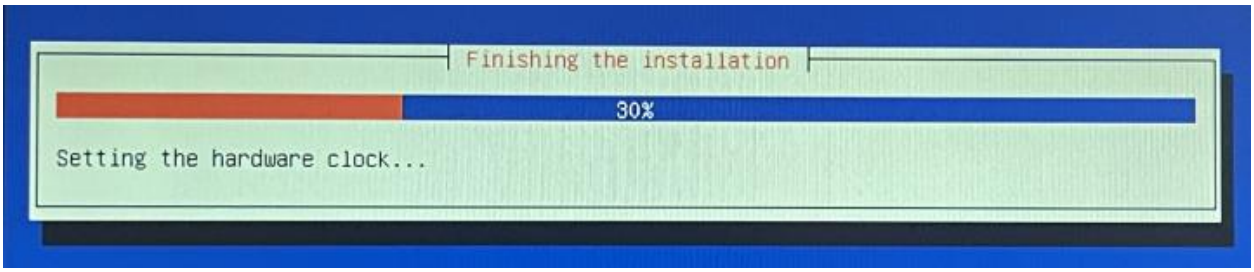


If this wasn't done, any time you'd run aptitude or apt-get to install or update packages, it would look for the `netinst` drive.

Debian does fully support physical media as a source - that's actually desired if say you downloaded the full distro ((all 10 or so DVDs) and wanted to install it on a system without an Internet connection.

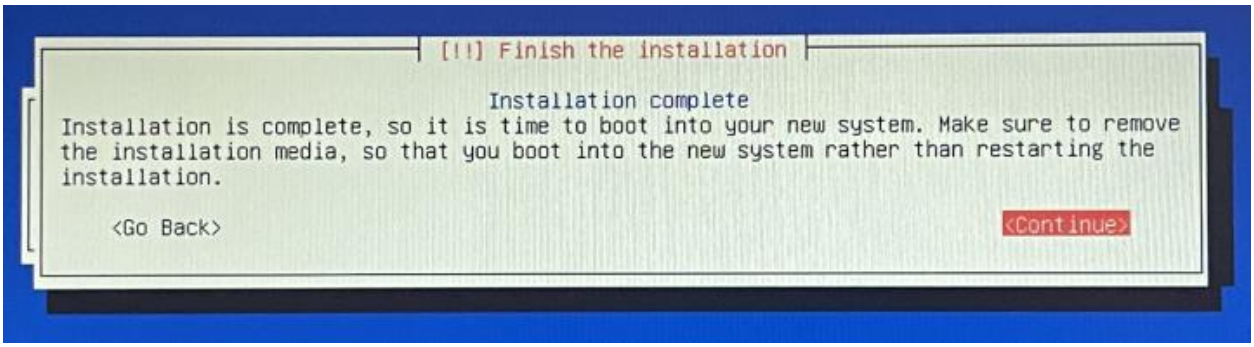
But in this case the Internet is our primary source, so the physical media is disabled as an installation source.

Then this screen …

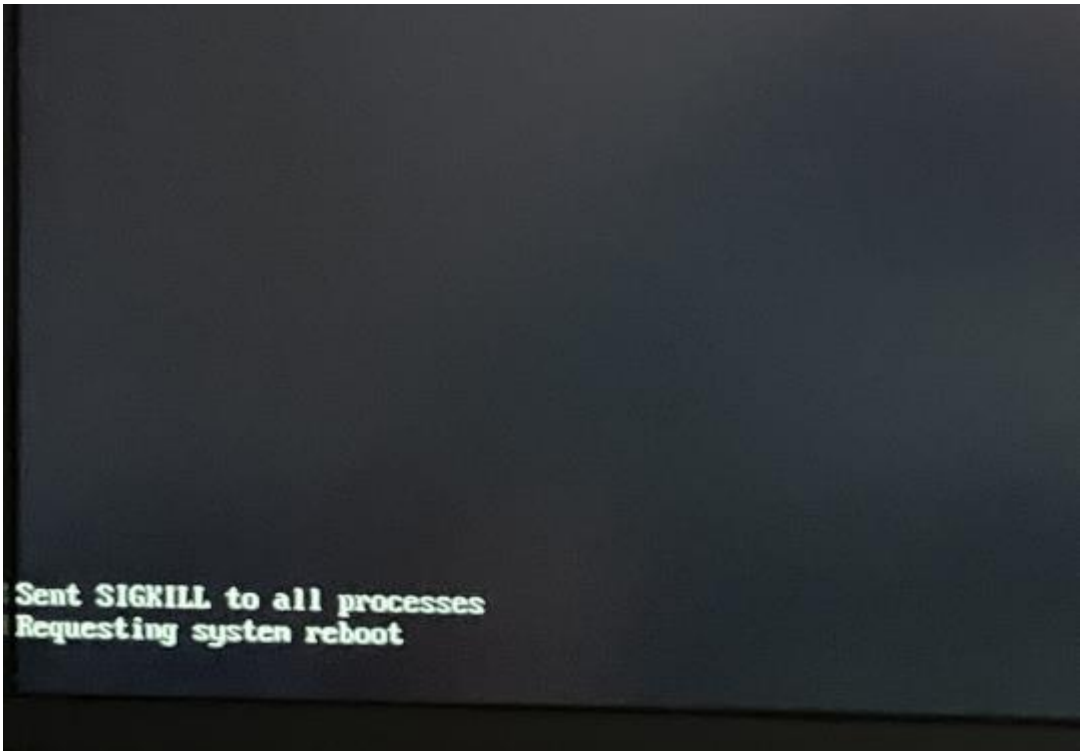


and then the LAST screen.

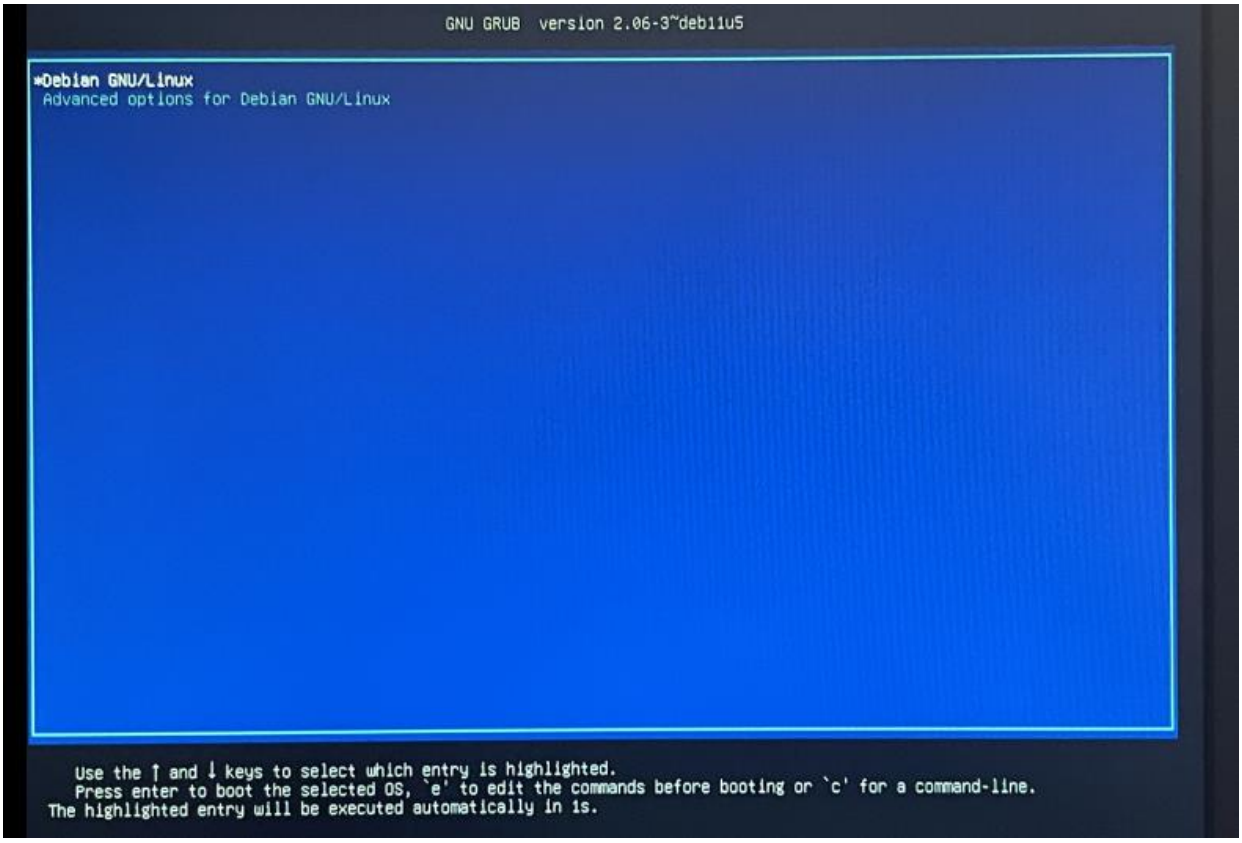
Remove the USB flash drive and hit "Continue."



This scary message will pop up and the system will reboot.



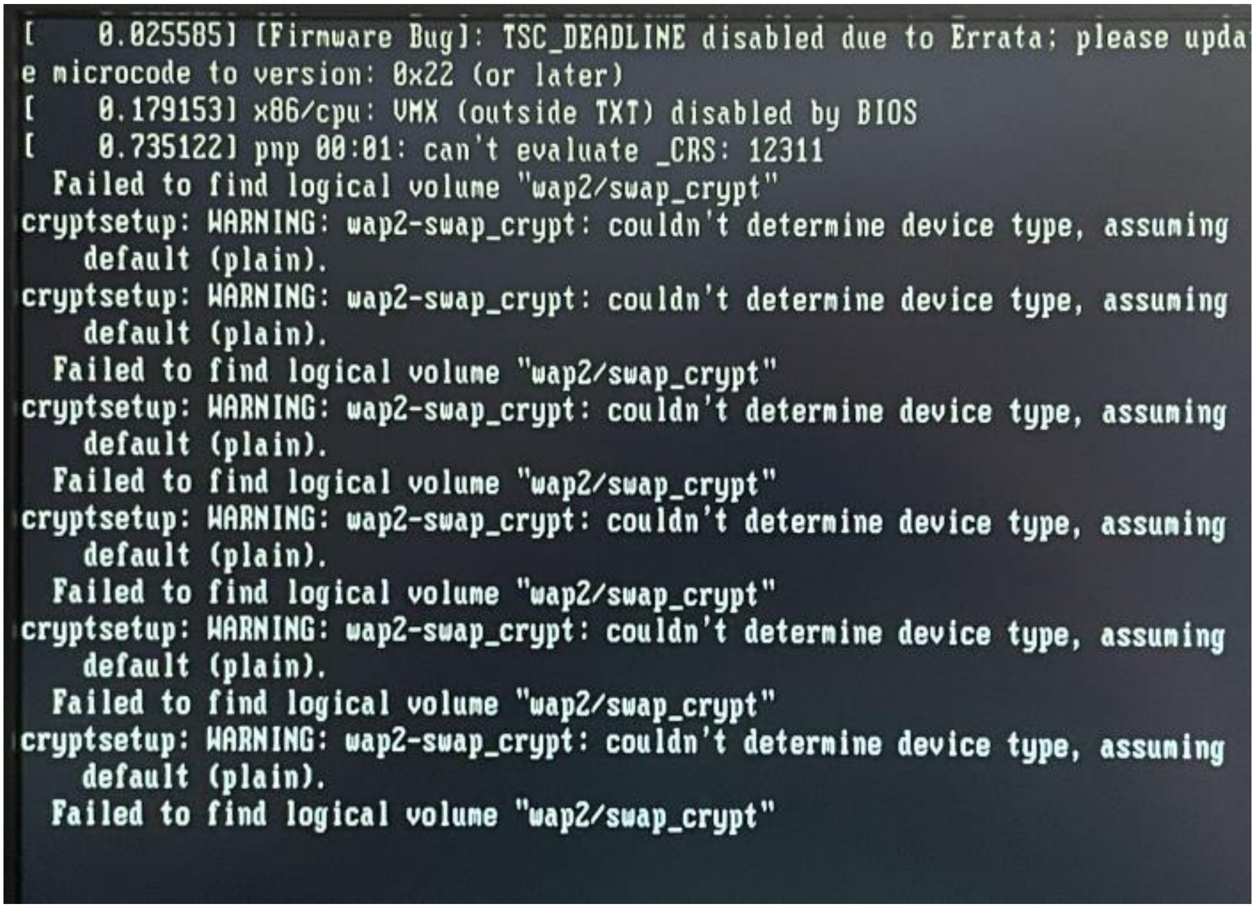
If everything went well, you should see the GRUB screen.



On Debian without a desktop environment installed, you're going to see some ugliness similar to the below. It is normal.

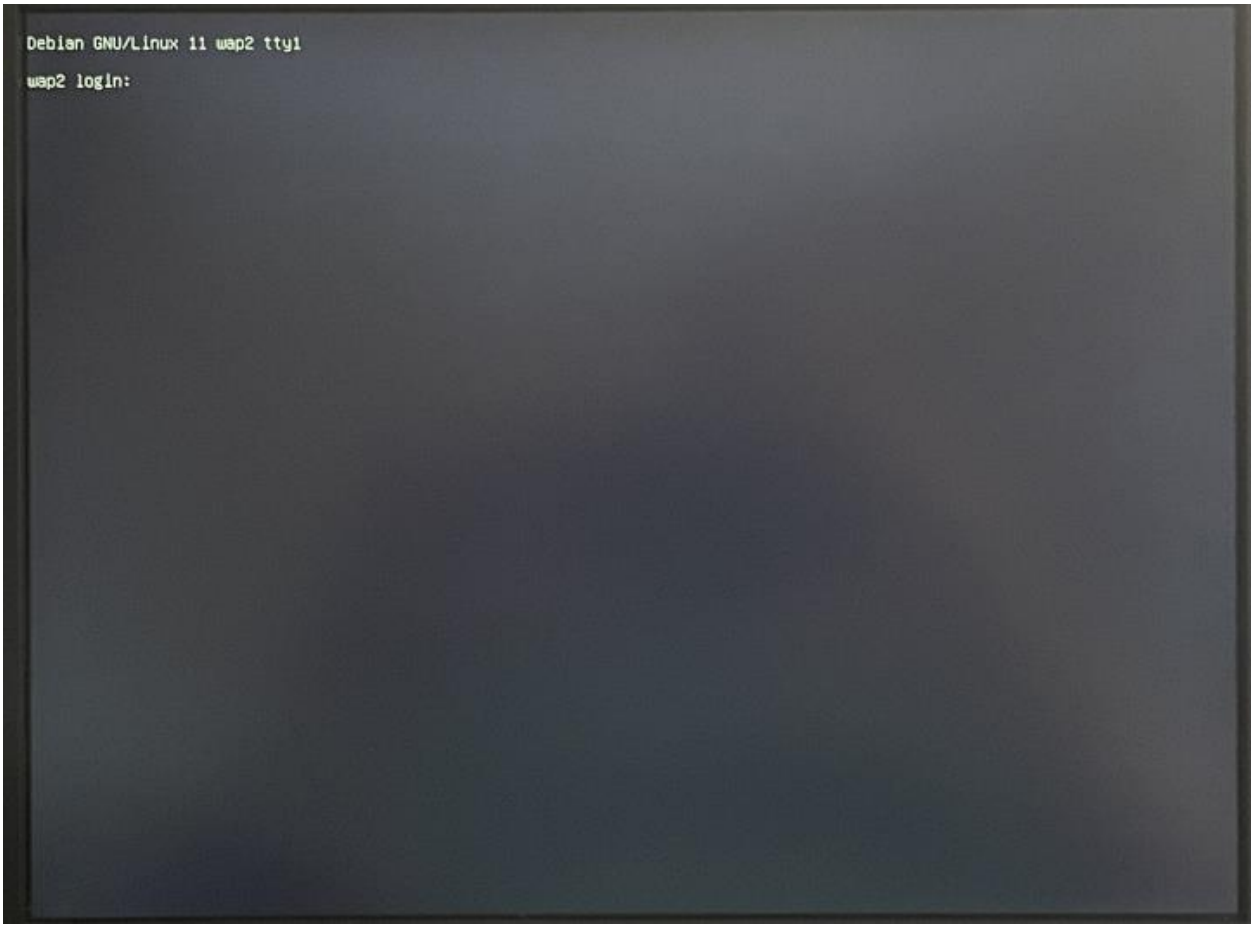
Firmware Bug and ACPI/pnp errors are normal. It's because PC firmware vendors suck.

The "failed to find" and "couldn't determine device type" messages aren't a concern—as `systemd` completes initialization and sets up the devices, they will be found and it will disappear. It may take 15 or so seconds.



All right. Ready to rock!

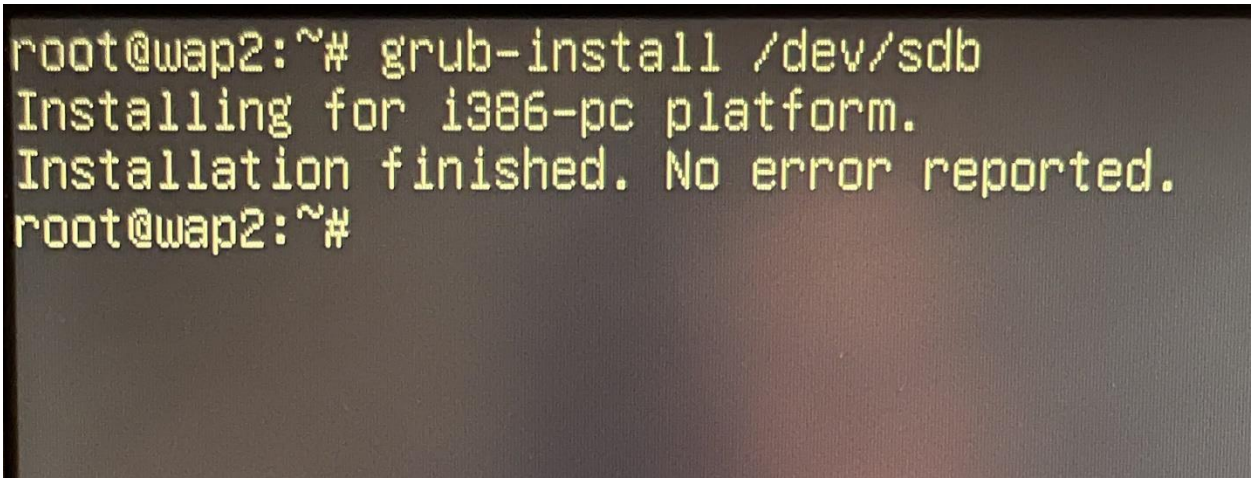
Just ONE more little step.



Making The Second SSD Bootable

Log in as `root` and run the following command:

grub-install /dev/sdb



Now, if the first drive dies, it'll boot off of the second one.

And that's it.

You now have a slim, basic Debian install operating off of a RAID-1 that will survive one hard drive failure, and can start customizing it according to your needs.