From Zero to Bootable: GSOS, My Own 'Shitty' Linux Distro

Guilherme Lima

July 16, 2025

G : Guilherme's
S : Shitty
O : Operating
S : System

What is this article about

This article was written to describe my experience while I created a Linux distribution, more precisely GSOSLinux (Shitty, not due to quality, but to lack of usability), following the Linux From Scratch guidebook. I had no previous experiences building kernel stuff from scratch other than your everyday ArchLinux installation. While the results aren't groundbreaking, neither is the system ready for daily driving, I do believe everyone who works with Linux systems should probably go through this experience at least once in their development career.

Who are you, and why should I care?

Hey there, my name is Guilherme, I'm a Brazilian CS major and I work as a software engineer intern. I started using Linux when I entered uni, around 2022. I was always a tech-savvy person, but this change made me realize how vast the world of technology is and that, despite how much knowledge we build as individuals over the years, we are incapable of even scratching the surface of what the people that came before us discovered in this field.

While I don't consider myself an expert in the area of systems engineering, I think this article will be a valuable read for people who haven't even dabbled in this sort of thing, are scared to break their computers, and think they are not capable of doing cool things on their machines without having to rely on high-level concepts.

Nowadays, the people in my circle are usually way less interested in low-level stuff, preferring things that are more popular, such as Artificial Intelligence or Web development. With this article, my main goal is to get more people interested not only in Linux, but also in knowing how their computers work, which for me, was like making something I thought was magic seem not so mystic anymore.

How it all started

This is my last year in university. Because of this, I had to choose a project to complete my curriculum. As someone who loves learning about Linux, I had to choose something along those lines. So that's what I did. The project will start this year in August and will involve tinkering with the kernel to achieve a more performant system. But that's not what this article is about. I'm here to tell you what I did to prepare myself for this project, the strategy I used to get to know more about the tool I will be working with for the next 12 months.

> "Linux From Scratch (LFS) is a project that provides you with step-by-step instructions for building your own custom Linux system, entirely from source code."[1]

I first came across this project as a joke. After switching distros a thousand times over and finally succeeding to install Arch for the first time. One of my friends showed me the website for LFS, saying that it was the next step - "If you think you know much about Linux, why not try building it yourself, easy enough right?" - That's definitely not what he said, but let's go with that for theatrical purposes. I really didn't think I was up to the challenge, I thought that was way out of my level, so I had to put the thing on hold for the time being, fearing not being able to accomplish the task. After messing with my operating system a bit more and motivating myself with my last uni project, I decided to pick it up again, and I'm glad I did.

Before doing it though, I faced a setback that would probably get many people reluctant to continue with such a project: My computer bricked. While on a trip abroad, I had the opportunity to buy an Apple computer, something which was way over my budget here in Brazil. While I did enjoy what MacOS had to offer (I'm not a Linux purist, neither am I a commercial system hater), some things I could do on Linux definitely left me longing for more control, and that was when I discovered Nix (calm down, I'm not here to ass pull Nix and say why everyone should start using it). Nix is awesome, it provides you with the flexibility of changing the software setup of your system faster than any other package manager can, but that's a talk for another time.

Now, back to our story, after discovering the wonders of Nixland and using its distro on my work computer, I decided to make my Mac more flexible and "Linux-like". Before that though, I wanted to factory restore it, because Nix

[1] Linux From Scratch

messes with your computer's privileged directories, and mixing between what's Nix and what's not is usually not a good idea. Apple had some things to say about what I had planned to do though. After selecting the restore button (using Apple's own restoring mechanisms), the screen just went black and it bricked on me. I tried fixing it myself (BTW, you need another Mac to fix a Mac) but to no avail. I had to go to Apple's support and make them figure that out. While I was worried that I might've lost my computer (which I'm sure I wouldn't be able to buy another one anytime soon), I knew that I would continue tinkering with computers in the future, and if it was not my fault, I shouldn't even bother worrying about it. Result: They just unplugged the physical battery and pushed it in again. Everything went back to normal after that.

The message I wanted to convey with this story is that you shouldn't be afraid to explore everything your machine has to offer. Sure, some casualties will come along the way (I lost a Nintendo DS because of that), but at the end of the day, the knowledge you're getting with these experiments are usually way more valuable than the setbacks you'll be facing.

The journey begins with flakes

I decided to go on with my LFS journey. The first thing you need to do is to have a base system to compile the first programs of your new system, to achieve a process called cross compiling (basically one machine M_a compiles programs that work on another machine, M_b, so M_b can then later use to compile things for itself natively), so I used my home PC (which was subject to many atrocities during my other little experiments) with a newly installed operating system for that purpose. For that, I did what many people in the community warn you against and tried using an unconventional Linux distribution, namely NixOS, as that base system.

That's a bad idea, don't do that. While Nix is great (I feel like I'm repeating myself here), it makes compiling and installing Linux software from scratch a way harder task, because the people who packaged these programs took some things for granted, things that Nix altered drastically to make their system more geared towards accomplishing its own objectives. During my first try, most of the installations failed due to some of those customizations, forcing me to start over in another "normal" distro. After trying a bit more, I decided to give up on that idea and switched to ArchLinux instead.

Continuing, the normal way

Many people avoid Arch. Sure, it's not as easy to install when comparing to the likes of other popular distros, such as Ubuntu or PopOS. Not having a graphical installer is something that instills fear into many. But, even though it's not noob-friendly, I think it is a necessary step into the Linux world. It's not as hard

as it seems, and you'll come out of it a lot more knowledgeable about how Linux works. Take a weekend and a spare old computer to try it out. I recommend this installation guide, it provides a comprehensive step-by-step that I think most people won't have to bash their heads against the wall to follow.

With that out of the way, let's go into more detail about the LFS side of things. As previously stated, the first part involves downloading a bunch of software to compile into your new system. This part is really tedious but straightforward. Just follow the guide and copy and paste the commands exactly as they are shown. One important thing I need to note is, don't rush through it. I know curiosity will usually get the best of us, making us want to achieve our expectations as soon as possible, take your time, read about what you are doing (this is the most important thing that LFS has to offer), and relax; it will take a lot of time to compile everything.

Problems will happen and daddy AI won't be there to help you

I don't hate AI, ML nor LLM's, I use it almost daily to help me solve my problems. But, I know it's not perfect and it definitely can't solve everything. People usually deposit a lot of trust into these complex linear algebra black boxes. One thing I know for sure is that it'll help you way more if you know what you're doing. AI has a problem that users tend to forget, it's very biased toward the prompter. If you ask it to build a house using a pillow for a hammer, it will try to do just that for the best of its abilities. That's what it was made for, after all.

Here, I was an (almost) complete amateur. The build processes vary from program to program and things are just hidden inside the build source code. Versions of the programs you are compiling matter, the version of what they depend on matter and, most importantly, the base system configuration in the machine you are using matters.

A specific problem I ran into was: while compiling Bash (a shell that allows the interaction between the user and the system), the build process failed. Following the most simple troubleshooting suggestion, I started again from scratch, at that point I had invested almost the entire day for that run. No worries, I tried again, and again, and again. Same problem. It was so frustrating, spending hours on a run that just amounted to nothing. Of course, things got faster and more mechanical once I was repeating the same thing again, but compilation takes time, and spending this time waiting for nothing makes you really stressed out.

I decided to dig into the error message and see what was going wrong. Copying the error into the AI just made things worse. It suggested making changes directly into some build flags and even changing the source code for Bash (don't do that, chances are you'll break the system later on). After trying all these

suggestions and failing every time, I took a break, then proceeded to search for the answers to my problems on the internet. After finding a possible solution in an obscure issue page for a random open source internet router firmware and the same thing in some underground tech forum, I just had to change the compiler flag to a specific standard `--std=gnu17`. As I tried to apply the changes... nothing. Make, the build tool used for most programs in LFS, shows each command that's being run. The compilation calls seemed to ignore my changes; I even reached a point where I changed all the compiler calls to use this standard, but it still ignored me. Long story short, Grep and Find became my best buds during this troubleshooting, I discovered that other build scripts were being run, and the changes I needed to apply was in one of those secondary scripts. And so, the lesson learned here was: Sometimes, the solution to your problems is not hanging in your face, so the more you understand what's happening in the background, even though that sometimes require a lot of time investment, the easier it becomes to get the answers you are looking for.

Another thing that I arrived at while having to repeat the same things again and again was: scripting exists and it saves you a ton of time. Ironically, Bash helped me on this one. It allows you to create scripts to automate commands using shell language, and it's very powerful, so use it to your heart's content. This repo holds some of the scripts I used to help me, they are not sure to work for you if you decide to embark in this journey, but they can serve as a base for your modifications to suit your needs. One important thing you need to take into account is to try to do things manually at first so you can learn; only then should you rely on automation, this will help you to debug problems as they happen and avoid breaking the system.

What I learned about Linux

Linux is not the whole operating system, compiling and configuring the kernel surely was not the most time-consuming part of the LFS guide. Programs are what makes the OS look and feel the way it does. That's where GNU enters the scene. I'm sure some of the people reading this dislike the man behind it, due to the controversies he's a part of. However, one thing is for sure, this organization provided an important step into making Linux as prevalent as it is today and, without them, I'm sure the scene of open source software would be very different.

Talking more about the importance of non-kernel software. Basic functionality like security, networking, identity management, console interfaces and more are the responsibility of these programs. Getting to know them and their importance was a huge stepping stone in discovering how my system works, and I will now be able to use them more directly to control my operating system more finely, making it act the way I want. But I'm still in this discovery stage, so I will avoid entering the territory I still know little of.

I did it, I have built Linux from scratch! What's next?

If you actually decided to go with it and succeeded in building your own system, congratulations! I'm sure you enjoyed this journey as much as I did and screamed your lungs out when you successfully booted into it. But what now? You have a barely working minimal system with no features of a daily driver whatsoever. Even though it seems like there was little concrete gain from this experience, I'd like you to remember that the journey was way more important then the destination. If you wish to continue working on that system, there's BLFS (Beyond Linux From Scratch). It'll help you shape your system into something more usable. As for me, I will continue with my curriculum project and hope to achieve some even greater results.

I hope you enjoyed the read and became more excited to give LFS a spin. If there's anything you'd like my help with, try contacting me via any social media platform, We're in this together and I'll try my best to help you out. That's all folks!