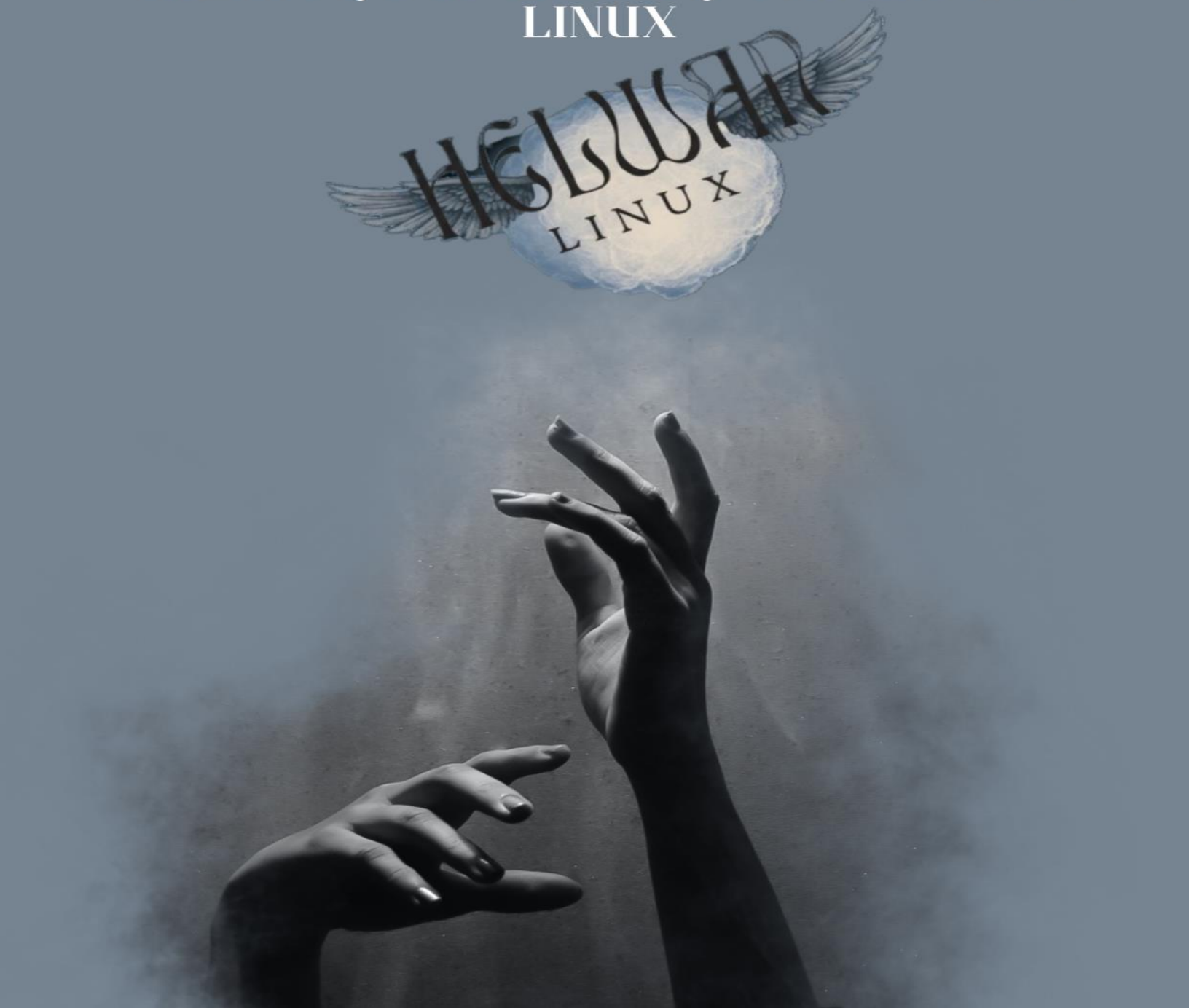
CUSTOM LINUX DISTRO BUILDING

A COMPREHENSIVE GUIDE TO ARCHISO, CALAMARES, AND HELWAN LINUX



About the Author

My name is **Saeed Mohamed Abdel Malek**, "Saeed Badreldin" though many in the tech community know me as **"S.M.A"**, after my YouTube channel *SMA Coding*. While my academic background lies in financial accounting, my true passion has always been technology. This passion constantly drove me to explore the world of computing and free software, always asking myself: *"How does this work?"* and *"Can I build something better?"*

That persistent curiosity led me deep into the world of operating systems, where I discovered the philosophy of **Arch Linux**—a philosophy that perfectly aligned with my own vision: simplicity, complete control, and the ability to build anything from scratch. From this principle, the idea of **Helwan Linux** was born. It was not merely a technical project, but a dream of creating a distribution that reflects the identity of the Arab community, delivers a smooth and personalized user experience, and stands as an inspiring example of what can be achieved with passion and knowledge.

Over the years, I have documented this journey on my channel *SMA Coding*, striving to share everything I learned with programmers and Linux enthusiasts everywhere. This book is the culmination of that journey: not just a technical manual, but the distilled essence of years of trial and error, learning, and sharing. My hope is that within these pages you will find everything you need to turn your ideas into reality—and that you will be inspired to play an active role in the open-source community.

Dedication

The price of this book, anywhere in the world, is to offer a sincere prayer for mercy upon my father and mother, regardless of your beliefs.

The true cost of this book is to share it as widely as possible.

Dedication

To my beloved father,

To my beloved mother,

To my sisters, the beat of my heart

To my loved ones:

the light of my eyes, my son Mohamed;

the joy of my heart, my son Omar;

and my precious daughter Malak.

You are the greatest inspiration in my life, and your presence constantly reminds me of the importance of building something worth passing on.

This book is for you, a token to show that passion and knowledge can create a better world.

Dedication

To the giants of technology who shaped our digital world:

- **Dennis Ritchie** and **Ken Thompson**, the spiritual fathers of the C language and Unix, who laid the foundation for everything we use today.
- **Richard Stallman**, the philosopher of free software, who sowed the seeds of the movement.
- **Linus Torvalds**, who made Linux a reality and united us under one banner.
- And to every developer and programmer working behind the scenes, staying up late to fix bugs and write code.

To the icons of this community:

- **The Ninjas**: developers who hide behind a black screen, coding silently and masterfully.
- **Hackers**: not destroyers, but explorers of systems, tracing vulnerabilities to understand and improve.
- **Contributors**: those who add small pieces that become the building blocks of massive projects, from a single line of code to translating a document.
- **Arch Linux users**: who embody the DIY (Do It Yourself) philosophy in their lives and believe in having full control over their systems.

This dedication is a token of gratitude to a community that believes in giving without limits.
May the flame of knowledge continue to burn brightly.

About the Book: A Roadmap to Digital Creation

Have you ever looked at a Linux operating system and imagined what it would be like if it were designed specifically to fit your unique taste and needs? Have you dreamed of owning a distribution that bears your personal signature, contains the tools you use every day, and meets the needs of your local community or professional world? If these ideas have ever crossed your mind, then this book is not just a guide; it's your companion on a journey to turn that dream into a tangible reality. It is an open invitation for every creator, developer, and operating system enthusiast to move beyond the limits of consumption into the world of production. Together, we will embark on an exciting journey that begins with the deep fundamentals of the Linux system, moves through exploring the technical secrets of the powerful tools used by professionals, and culminates in building your own project from scratch.

What will the pages of this book offer you?

This book goes beyond being a mere collection of technical instructions; it is a strategic roadmap that enables you to dive into the depths of the open Linux world and gives you the tools for complete control. With each chapter, you'll acquire advanced skills that put you in the driver's seat:

- **A Deep and Philosophical Understanding of Arch Linux:** You won't just learn how to use Arch; you'll understand the philosophy it's built on. You'll discover the principle of "complete control" and how this solid foundation can serve as a blank canvas for your project, freeing you from the constraints imposed by other distributions.
- **Mastering Archiso from Zero to Professionalism:** This part is the heart of the book. You'll transform from a regular user into a "release engineer." You'll learn how to build bootable ISO images and meticulously customize every aspect of the live system, including adding packages, choosing desktop environments, and writing custom scripts that automate your tasks.
- **Complete Control over Calamares:** You will understand how to integrate a smooth and professional graphical installation interface into your distribution. You'll delve into customizing every detail of Calamares, from designing the user interface to modifying installation modules to ensure a unique installation experience that perfectly aligns with your vision.
- **Turning Ideas into Reality: Building Your Own Distribution:** After mastering the tools, you'll have the knowledge and skills needed to transform your abstract ideas into a practical, stable, and ready-to-use Linux distribution. You'll go beyond simply adding packages to reach the stage of building a comprehensive and interconnected system.
- **Drawing Inspiration from the Helwan Linux Experience:** We will provide you with a practical and inspiring model from the heart of the Arab community. The Helwan Linux experience is a living testament that passion and knowledge can produce a powerful and useful project. By reviewing this project, you will gain practical insights into how to assemble all the pieces together and how a small community can build something of great impact.

For Whom Is This Guide?

This book is not restricted to a specific category but has been carefully designed to serve as a bridge connecting different levels of expertise in the world of Linux:

- **For the Ambitious Beginner:** If you've just started your journey in the world of Linux, this book will guide you step by step. We'll begin by explaining complex concepts in a simplified manner, focusing on practical examples that enable you to successfully build your first project.
- **For the Intermediate User:** If you're familiar with Linux fundamentals and use distributions like Arch or others, this book will open new horizons for control and customization. You'll move from just using the system to understanding how to build and modify it from the inside.
- **For the Advanced Developer:** If you're looking to create ready-made development environments, build custom tools for security or scientific purposes, or even launch a Linux distribution for your commercial or community project, you'll find here the advanced resources and examples you need to execute your ideas with the highest efficiency.

Arch Linux: The Cornerstone for Ambitious Projects

Arch Linux was not chosen as the base for this book randomly; it's a strategic decision based on several fundamental principles that have made it the optimal choice for custom projects:

- **The KISS Principle (Keep It Simple, Stupid):** Arch adopts the principle of "keep it simple." This means it avoids unnecessary complexity, making it transparent and easy to understand how it works from start to finish. You build your system yourself and control every part of it.
- **Absolute Customization Control:** Unlike distributions that come with pre-selected desktop environments and programs, Arch gives you absolute freedom. You decide what's in your system, allowing you to build a lightweight, efficient, and flexible system that precisely meets your needs.
- **Rolling Release Model:** You get the latest software versions instantly, ensuring your distribution is always up-to-date and benefiting from the newest features and security fixes as soon as they become available.
- **Unmatched Documentation and a Collaborative Community:** No other Linux system has official documentation (Arch Wiki) with this level of detail and accuracy. Alongside a broad and collaborative community, you'll always find the help and information you need to overcome any challenge.

Helwan Linux: More Than Just a Distribution

At the heart of this guide, we don't just offer you tools; we inspire you through a living, practical example. **Helwan Linux** is a pioneering experience of an ambitious Arab project built on the philosophy of **Arch Linux**, aiming to provide a smooth and customized Linux experience for users in the Arab region. By analyzing this project, you'll learn how to apply the concepts we'll be learning, and how collective passion can create something with real impact. Helwan Linux is not just a distribution; it's a living testament that creativity knows no bounds and that knowledge can be transformed into powerful and beneficial community projects.

Tips for Getting the Most Out of This Journey

Theoretical knowledge alone is not enough. To get the most out of this book, we invite you to adopt the mindset of a researcher and an experimenter:

- **Practical Application is Your Key:** Don't just read. Create a virtual environment and start applying every step; practical experience is the best teacher.
- **The Arch Wiki is Your Best Friend:** Don't hesitate to refer to the **Arch Wiki** for additional details on any package or command.
- **Be Curious and Adventurous:** Change settings, add new programs, and try to understand what's happening behind the scenes.
- **Community Participation:** Share your questions and experiences in Arab and international technical forums. Knowledge grows through sharing.
- **Regular Review:** Go back to previous chapters whenever you feel the need to solidify concepts or gain a deeper understanding.

We wish you a fun and fruitful journey into the world of building Linux distributions, and we hope these pages will be your launchpad toward limitless digital creations.

Chapter One: Fundamentals of Arch Linux and the Working Environment

Section 1: Introduction to Arch Linux

1.1 A Historical Overview of Linux and the Rise of Arch

1.1.1 The Genesis of the Linux System (1991)

In the early nineties, the world of operating systems was dominated by commercial **Unix** and Microsoft's **MS-DOS**. Unix systems were powerful but very expensive and closed-source, which limited their accessibility to only universities or large corporations.

In 1991, **Linus Torvalds**, a computer science student at the University of Helsinki in Finland, wasn't satisfied with the **Minix** operating system (a small version of Unix for educational purposes). He decided to write his own kernel as a hobby and released it online under an open-source license.

In his first public announcement, Torvalds wrote: "I'm working on a (free) operating system as a hobby, it won't be anything big or professional like GNU."

But soon, hundreds of programmers from around the world joined to develop this new project, which was named the **Linux Kernel**. When it was combined with the tools from the **GNU** project (founded by Richard Stallman), we got what we know today as the **GNU/Linux** operating system.

1.1.2 The Beginning of Distributions

Since Linux was just a kernel without ready-to-use tools, people needed an easy way to bundle the kernel with essential programs and distribute it as a complete system. This is how the idea of a **Distribution** was born.

Among the first distributions were:

- **Slackware (1993):** One of the oldest distributions, still around today.
- **Debian (1993):** Focused on stability and an organized community. It later became the foundation for many distributions like Ubuntu.
- **Red Hat Linux (1995):** Geared towards businesses, it later became the basis for RHEL and Fedora.

Each distribution had its own philosophy: some focused on ease of use (like Ubuntu later on), some on stability (like Debian), and others on customization (like Gentoo).

1.1.3 The Emergence of Arch Linux (2002)

In the early 2000s, a Canadian developer named **Judd Vinet** noticed that most existing distributions were either too complex (like Gentoo, which required building the system from source) or "bloated" because they came preloaded with settings that not every user needed.

In 2002, Judd Vinet decided to create a new distribution he named **Arch Linux**.

- **His goal:** A simple, lightweight, and flexible distribution.
- **Its motto: KISS** (Keep It Simple, Stupid).
- It didn't come with a graphical interface; it started from the command line.
- It used a new package manager called **pacman**, which was easy and practical for managing packages.

Arch Linux was not aimed at beginners, but it attracted a community of advanced users who loved the freedom of customization and the complete transparency of the system.

1.1.4 The Community Takes Over Development - Aaron Griffin

After a few years of leading the project, Judd Vinet decided to stop developing Arch Linux for personal reasons. In 2007, **Aaron Griffin** took over the leadership.

During Griffin's tenure:

- Arch's development became more organized and community-driven.
- The **Arch Wiki** flourished and became one of the most important knowledge sources in the Linux world.
- Derivative projects of Arch, such as ArchBang and Manjaro, emerged.

Today, Arch Linux is managed by a team of core developers and a huge community of contributors, which has made it one of the most powerful and influential Linux distributions.

 Summary of This Section:

- Linux started as a hobby for a university student in 1991.
- It quickly became a global system thanks to the open-source philosophy.
- Distributions emerged to make it easier to use, each with its own philosophy.
- Arch Linux (2002) came as a middle ground: not as complex as Gentoo, nor as restrictive as Ubuntu.
- Today, Arch is not just a distribution, but a school that teaches the user how the system works from the inside out.

What Makes Arch Linux Unique?

When we talk about Linux distributions, terms like "ease of use," "stability," or "technical support" often come to mind. But with **Arch Linux**, the situation is different; it's not just another operating system, but a complete philosophy for interacting with computers. To understand what distinguishes Arch, we must look at several key areas:

1.2.1 Simplicity

Arch's core motto is **KISS** (**K**eep **I**t **S**imple, **S**tupid).

However, the word "simplicity" here doesn't mean the system is easy for beginners to use. It means:

- Avoiding unnecessary design complexity.
- Not adding unnecessary tools or interfaces.
- Providing only the bare minimum and leaving the rest to the user.

For example, distributions like **Ubuntu** come pre-installed with the GNOME interface, Firefox browser, and the LibreOffice suite. In contrast, when you first install Arch, it gives you an almost empty **base system**. You decide: which graphical interface do you want? Which browser? Which office tools? This makes Arch more like **Lego bricks**: you build the system piece by piece according to your wishes.

1.2.2 Transparency

Arch hides nothing from the user:

- All configuration files exist as readable and editable text.
- There's no "magic" behind the scenes.
- Even the installation process itself is a series of commands the user types, allowing them to understand how the operating system is built step-by-step.

For example, in Ubuntu, a user can add a software repository through a graphical interface or a simple command like `add-apt-repository`. In Arch, this is done manually by editing the `/etc/pacman.conf` file, which makes you see and understand exactly what's happening.

1.2.3 Flexibility

Arch doesn't impose any decisions on you:

- Want to use KDE Plasma? You can install it.
- Prefer GNOME or XFCE? Also possible.
- Don't want a graphical interface at all and prefer a simple window manager like `i3wm` or `bspwm`? It's fully available.

This flexibility makes Arch suitable for various users, from lovers of speed and a minimal footprint to enthusiasts of advanced aesthetics and interfaces.

1.2.4 Rolling Release

One of the biggest differences between Arch and other distributions is the update system:

- Most distributions (like Ubuntu or Fedora) release new versions every 6 months to a year. The user needs to upgrade between releases.
- Arch, however, is a **Rolling Release**, meaning the distribution is always up-to-date. Installing Arch once is enough for years; you just need to keep it updated via `sudo pacman -Syu`.

Advantages:

- You always get the latest versions of software and the kernel.
- You don't need to reinstall the system.

Disadvantages:

- An incompatible update can sometimes cause sudden problems.
- The user needs to pay attention to **Arch News** before updating.

1.2.5 The Pacman Package Manager

One of the pillars of Arch Linux is the **pacman** package manager.

- It's designed to be simple and fast.
- It automatically handles dependencies.
- Commands are unified and easy to remember.

Common Examples:

Bash

```
# Update the system
sudo pacman -Syu

# Install a program
sudo pacman -S firefox

# Remove a program with its extra files
sudo pacman -Rns firefox

# Search for a program
pacman -Ss vlc
```

In comparison:

- `apt` in Debian/Ubuntu can require more complex commands.
- `dnf` in Fedora is sometimes slower in performance.

1.2.6 Arch Wiki: The Encyclopedia of Knowledge

One of the most important things that distinguishes Arch is not the distribution itself, but its documentation.

- **The Arch Wiki** is considered one of the largest and most comprehensive Linux references ever.
- Even users of other distributions (Debian, Fedora, Manjaro) rely on it to solve problems.
- It explains everything, from installing a graphics card to setting up a complete web server.

1.2.7 AUR (Arch User Repository)

Arch has huge official repositories, but its true power shines with the **AUR**:

- It is a repository built entirely by the community.
- It contains thousands of packages that you won't find in the official repositories.
- It's managed via **PKGBUILD** files that allow you to build the package on your machine.

For example, a program not officially available, like Google Chrome, can be easily found in the AUR via a tool like `yay`:

Bash
```
yay -S google-chrome
```

But you must be cautious:

- Not all packages in the AUR are guaranteed to be of high quality or secure.
- It's always recommended to read the **PKGBUILD** file before installation.

1.2.8 Arch as a Base for Other Distributions

Arch's power has made it the foundation for a number of derivative distributions:

- **Manjaro:** Offers Arch with a ready-to-use graphical interface for beginners.
- **EndeavourOS:** Provides an experience closer to Arch but with an easier installation.
- **Garuda Linux:** Focuses on graphics and performance.
- **Helwan Linux (our example):** An Egyptian distribution built on Arch with a local flavor and a unique identity.

This proves that Arch is not just an operating system, but a platform for building other systems.

 Summary of Section 1.2:

What distinguishes Arch is not just the technology it's built on, but the philosophy it adopts:

- **Simplicity**,
- **Transparency**,
- **Flexibility**,
- **Rolling Release**,
- The power of the community,
- And relying on the user as a core element in building their experience.

1.3 The Philosophy of Arch Linux in Depth

Arch Linux is not just a lightweight or flexible distribution; it's a complete school in the philosophy of operating system design. If we want to understand the "soul" of Arch, we must dive deeper into the principles that guide it.

1.3.1 The KISS Principle (Keep It Simple, Stupid)

The "simplicity" here is not about ease of use. Arch does not try to be easy for beginners like Ubuntu or Mint. Rather, it means **clarity and lack of complexity**: there are no hidden layers of middleware or automatic management tools that impose themselves.

For example:

- In Ubuntu, when you install a graphics card driver, there are dedicated graphical tools that perform the task for you.
- In Arch, this is done by manually installing the appropriate packages from `pacman` or the AUR and editing the specific configuration files.

The result: the system is simpler from the inside, but requires more knowledge and experience from the user.

1.3.2 Complete Control (User Centrality)

Arch Linux is built on the idea that the user knows their needs best.

- The system does not impose specific packages or settings on you.
- Even the installation process does not provide a graphical interface but gives you the basic tools to build the system yourself.

This complete control makes Arch ideal for developers and engineers who need a customized work environment. For example, in Ubuntu, the system installs with GNOME by default. In Arch, after the initial installation, you don't even have a graphical interface; you have a black screen (TTY). If you want GNOME, you install it yourself; if you want KDE, XFCE, or even no interface, the decision is yours alone.

1.3.3 Transparency

One of Arch's core principles is that everything should be clear and understandable.

- There is no "magic" in the background.
- All settings can be adjusted via simple text files.
- The documentation (**Arch Wiki**) explains every step in detail.

For example, if you want to run a service in Arch, you use `systemctl enable...` and you understand what happens in the background. Whereas in other distributions, you might click a button in a graphical interface and not know what happened behind the scenes.

1.3.4 Learning by Doing

Arch Linux is different from most distributions because it teaches you as you use it:

- The installation process itself is a practical lesson in how Linux works.
- Network settings, users, and the boot manager are all steps the user goes through and understands.
- Over time, an Arch user transforms from a mere "consumer" of the system into a "controller" of it.

Some users describe the experience as: "Arch doesn't give you a fish, it teaches you how to fish."

1.3.5 Rolling Release as Part of the Philosophy

The choice of a rolling release system for Arch was not a random decision but stemmed from its philosophy:

- Why force the user to reinstall the system every 6 months or a year, as in Ubuntu?
- Instead, let the user get the latest software instantly, and their system continues to evolve with them.

This reflects Arch's belief in the idea that a system should always be alive, never growing old or outdated.

1.3.6 Community Driven

Arch does not seek to please corporations or a commercial direction. On the contrary:

- The community is the beating heart of the distribution.
- Most solutions are found in the **Arch Wiki**, which is written by users.
- The AUR repository is built entirely from community contributions.

This reflects an important philosophical principle: knowledge is collective and not exclusive to an institution or company.

1.3.7 Arch Is Not for Everyone (Selective Philosophy)

Arch's founder did not intend for the distribution to be easy or suitable for everyone.

- If you're a complete beginner, you might find Arch very difficult.
- But if you want to understand Linux from the inside out and control it, you'll find Arch is the best choice.

This principle has led Arch to be sometimes known as an "elite" distribution, not in a condescending sense, but in the sense that it requires a certain level of seriousness and curiosity.

 Summary of Section 1.3:

The philosophy of Arch Linux can be summarized in:

- **Simplicity** of design without unnecessary complexity.
- **Complete control** over the system.
- **Transparency** in everything.
- **Learning by doing**.
- **Continuous updates** that keep up with the times.
- A strong **community** that drives development and support.

This philosophy has made Arch more than just a distribution: it is a way of thinking and dealing with operating systems.

1.4 Rolling Release

One of the most notable features of **Arch Linux**, and what makes it different from most other distributions, is its continuous update system, known as **Rolling Release**. This concept isn't just a way to distribute software; it's a fundamental part of Arch's philosophy.

1.4.1 What Is a Rolling Release?

In the world of software, there are two main ways to distribute versions:

1. **Fixed Release:**
    - The distribution releases a new version at fixed intervals (e.g., Ubuntu every 6 months, Debian every two years).
    - The software within that version generally stays the same, with the exception of security updates.
    - **Example:** Ubuntu 22.04 will continue to use the same GNOME and kernel versions until the release of 24.04.
2. **Rolling Release:**
    - The system doesn't have "major versions." Instead, packages are continuously updated.
    - When a new version of the kernel or a program is released, it's updated immediately in the repositories.
    - **Example:** Arch Linux always has the latest version of the Linux Kernel, without waiting for a "new version of Arch."

1.4.2 Advantages of a Rolling Release

- **Always the Latest Software:** You get the latest version of your favorite text editor or coding IDE as soon as it's released. This is very suitable for developers who need a constantly modern environment.
- **No Need for Reinstallation:** With fixed-release distributions, you might have to reinstall or upgrade the system every so often. With Arch, one installation is enough for years; you just keep updating.
- **A System That's Always Alive:** Arch doesn't grow old. As long as you update it, it will always be in its latest form.

1.4.3 Disadvantages of a Rolling Release

- **Potential for Breakage:** An incompatible update or a bug in a package can cause a problem. For instance, a graphics card driver update can sometimes cause booting issues.
- **More Responsibility on the User:** You must continuously follow the **Arch News** for important alerts before updating and be prepared to fix problems yourself.
- **More Internet Consumption:** You are always downloading new versions.

1.4.4 How Arch Handles a Rolling Release

Despite the disadvantages, Arch has a robust system to minimize problems:

- **Package Testing:** Before any package enters the official repository, it's tested in a dedicated `[testing]` repository.
- **Documentation:** Any major problem is immediately documented on the **Arch Wiki** or the news page.

- **Community Support:** Users quickly share solutions on forums or Reddit.

1.4.5 Security Strategies for Updating

- **Update Regularly:** Not updating for long periods can make the system difficult to upgrade due to many conflicts. It's better to update weekly or every two weeks.
- **Use Timeshift or Snapshots:** You can take a backup before updating to roll back if a problem occurs.
- **Read the News Before You Update:** The command `sudo pacman -Syu` seems simple, but you should be aware of what might change after it.

1.4.6 Comparison with Other Distributions

Criterion	Arch (Rolling)	Ubuntu/Debian (Fixed)	Fedora (Semi-Rolling)
Kernel Updates	Always the latest version	Stable throughout the release's life	Relatively quick updates
System Stability	Less (but flexible)	Higher (suitable for servers)	Medium
Time Between Releases	None (continuous)	6 months or more	~6 months
Ease of Management	Needs constant attention	Easier for beginners	Medium

1.4.7 Practical Examples

1. **Game Developer:** Needs the latest Vulkan and Mesa libraries → Arch is ideal.
2. **Server Company:** Needs long-term stability without surprises → Debian or RHEL is better.
3. **Regular User:** Wants their device always working without worry → Manjaro (built on Arch but more stable).

□ Summary of Section 1.4:

The rolling release in Arch Linux is a powerful feature that makes it always modern and up-to-date. But it's a double-edged sword: it gives you freedom and modernity but requires responsibility and vigilance.

1.5 The Pacman Package Manager

One of the fundamental pillars of **Arch Linux**, and what distinguishes it from other distributions, is the **Pacman** package manager.

In Linux systems, the package manager is the tool that allows you to download, install, update, and remove software from a distribution's repositories. But Pacman stands out for its simple philosophy and powerful performance.

1.5.1 What Is Pacman?

- The name "Pacman" is short for **Pac**kage **Man**ager.
- It is the official tool for managing packages in Arch Linux.
- It's written in the **C** language to be fast and lightweight.
- It manages packages with a `.pkg.tar.zst` extension (compressed files that contain programs).

1.5.2 Advantages of Pacman

1. **Simple Commands:**
    - Commands are short and easy to remember.
    - For example: `-S` for installation, `-R` for removal, `-Q` for querying.
2. **Automatic Dependency Management:**
    - If a program needs additional libraries, Pacman installs them automatically.
3. **Speed:**
    - Thanks to its design in the C language and a repository system based on compressed files.
4. **Uniformity:**
    - The same tool is used for everything (installation, updating, searching, removal).

1.5.3 Basic Pacman Commands

Command	Function	Example
`pacman -S package`	Install a program	`pacman -S firefox`
`pacman -R package`	Remove a program	`pacman -R vlc`
`pacman -Rns package`	Remove a program + unused dependencies	`pacman -Rns gimp`
`pacman -Ss keyword`	Search for a program in repositories	`pacman -Ss vlc`
`pacman -Qs keyword`	Search for a locally installed program	`pacman -Qs python`
`pacman -Qi package`	Display information about an installed program	`pacman -Qi nano`
`pacman -Syu`	Update the entire system	`pacman -Syu`

1.5.4 Pacman's Configuration Files

- **Main file:** `/etc/pacman.conf`
    - Contains settings like enabled repositories, installation options, and digital signature management.
- **Local database:** `/var/lib/pacman/`
    - Where Pacman stores information about installed packages.

Example from `pacman.conf`:

```
[options]
HoldPkg = pacman glibc
Architecture = auto
CheckSpace
SigLevel = Required DatabaseOptional

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

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

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

1.5.5 Digital Signatures (Package Signing)

- Pacman uses the GPG system to verify packages.
- This means every package has a digital signature to ensure it hasn't been changed during transfer.
- If there's an authentication error, Pacman won't allow installation unless you force the system (which is not recommended).

1.5.6 Comparing Pacman with Other Package Managers

Feature	**Pacman (Arch)**	**APT (Debian/Ubuntu)**	**DNF (Fedora)**
Programming Language	C (very fast)	C++	Python/C
Command Difficulty	Easy and simple	Medium	Medium
Dependencies	Strong management	Strong	Strong
Updates	Always Rolling	Fixed according to versions	Semi-annually
Speed	Faster performance	Relatively slower	Slower than Pacman

1.5.7 Using Pacman with Additional Repositories

You can add extra repositories by editing the `/etc/pacman.conf` file.

Example: Adding the `multilib` repository (to run 32-bit programs):

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

Then, run: `sudo pacman -Syu`

1.5.8 Common Problems with Pacman and Their Solutions

- **Corrupted Database:**
  - Use `sudo pacman -Syy` (to force a database refresh).
- **Conflicting Files During Update:**
  - **Solution:** Manually remove the file or use the `--overwrite` option.
- **Internet Interruption During Update:**
  - The process can be resumed easily after reconnecting.

1.5.9 The Relationship Between Pacman and the AUR

While Pacman is very powerful, it doesn't directly manage the **AUR (Arch User Repository)**.

- For AUR packages, you use helper tools like **yay** or **paru**.
- These tools ultimately rely on Pacman but add the step of creating the package from a `PKGBUILD` file.

 Summary of Section 1.5:

Pacman isn't just a regular package manager; it's the heart of the Arch Linux experience. Its simplicity, speed, and flexibility make it one of the fastest and most powerful package managers in the Linux world.

1.6 The AUR (Arch User Repository)

One of the most prominent strengths of **Arch Linux** is the **AUR**, which is considered one of the largest community repositories in the Linux world.

1.6.1 What Is the AUR?

- **AUR** stands for **Arch User Repository**.
- It's a massive repository containing **package recipes (PKGBUILDs)** written by the community.

- Its goal is to allow users to easily install software not available in the official repositories.

Official repositories contain only software that the Arch team officially tests. The **AUR**, on the other hand, is an open space where users can share any program, tool, or even theme.

1.6.2 What is a PKGBUILD File?

- A **PKGBUILD** file is a text script written in **Bash**.
- It contains instructions for building a package from source or pre-compiled files.
- **Pacman** doesn't handle the AUR directly; the user builds the packages themselves using this file.

A simple example:

Bash
```
pkgname=hello
pkgver=1.0
pkgrel=1
arch=('x86_64')
source=("http://example.com/$pkgname-$pkgver.tar.gz")
md5sums=('SKIP')

build() {
  cd "$srcdir/$pkgname-$pkgver"
  ./configure --prefix=/usr
  make
}

package() {
  cd "$srcdir/$pkgname-$pkgver"
  make DESTDIR="$pkgdir/" install
}
```

1.6.3 Why Is the AUR Important?

1. **Broad Coverage:** Any program you can think of can likely be found in the AUR.
2. **Massive Community:** Thousands of contributors upload and update packages daily.
3. **Flexibility:** You can modify the **PKGBUILD** file yourself before building.
4. **Speed of Availability:** Programs are often uploaded to the AUR before they reach the official repositories (if they ever do).

1.6.4 How the AUR Works in Practice

1. You search for the package on the AUR website:
    - **https://aur.archlinux.org**
2. You copy the `PKGBUILD` file.
3. You use the command `makepkg -si` to build and install the package locally.

1.6.5 AUR Helper Tools

Since manually dealing with `PKGBUILDs` is tedious, the community created tools to simplify the process.

The most popular ones are:

Tool	Feature
yay	The most common helper; it works with both Pacman and the AUR.
paru	Similar to yay but with a simpler interface.
trizen	Supports advanced search and build features.
pamac	A graphical user interface (GUI) similar to the package manager in Manjaro.

Example using yay: `yay -S google-chrome` This command will search the AUR, build the package, and install it automatically.

1.6.6 Challenges and Risks with the AUR

- **Security:** Since the packages are written by the community, they might contain malicious code.
    - **Solution:** Always inspect the `PKGBUILD` file before building.
- **Package Quality:** Not all packages in the AUR are up-to-date or stable.
- **Dependencies:** Sometimes, there are dependencies that aren't available in the official repositories.

1.6.7 The Relationship Between the AUR and Official Repositories

- An AUR package can later be adopted and moved to the official repositories.
- **Example:** Many programs started their life in the AUR and became official after gaining popularity.

1.6.8 Comparing the AUR to Similar Repositories in Other Distributions

Distribution	Equivalent System	Comparison to AUR
Ubuntu/Debian	PPA (Personal Package Archives)	Similar, but much smaller in size and content.
Fedora	COPR	Very similar, but less widespread.
openSUSE	OBS (Open Build Service)	A comprehensive system but more complex than the AUR.

1.6.9 Examples of Famous AUR Packages

- **`google-chrome`:** Not officially available due to licensing.
- **`spotify`:** Closed-source, but available via the AUR.
- **`visual-studio-code-bin`:** The official pre-compiled version of Microsoft VSCode.

- **`whatsapp-nativefier`:** To convert WhatsApp into a desktop application.

 Summary of Section 1.6:

The **AUR** is the second pillar after **Pacman** that makes **Arch Linux** a unique system. Thanks to this repository, the user has access to thousands of unofficial additional programs, which opens the door to immense freedom and flexibility, but it requires responsibility and careful inspection from the user.

1.7 Arch Linux Philosophy: The KISS Principle (Keep It Simple, Stupid)

One of the most important pillars of Arch Linux is the **KISS** philosophy, which stands for: **"Keep It Simple, Stupid."**

While the phrase might sound sarcastic at first, it holds a deep vision for building software systems: simplicity is more powerful than complexity.

1.7.1 What Does Simplicity Mean in Arch?

- Simplicity doesn't mean a lack of features.
- Instead, it means the system is built on small, clear, and understandable components.
- Arch doesn't try to hide complexity from users (as distributions like Ubuntu or Fedora do); it puts the tools in front of you and gives you complete control.

1.7.2 Manifestations of the KISS Principle in Arch Linux

1. **Text Configuration Files:** There are no complex graphical tools to change settings. Almost everything is done through readable and understandable text files (e.g., `systemd`, `pacman.conf`).
2. **Package Management (Pacman):** A single powerful tool handles installation, updating, and removal. There aren't dozens of different tools like in some other distributions.
3. **Focus on the Core:** Arch provides you with a clean base system. The rest is up to you to build step-by-step.
4. **Documentation (Arch Wiki):** Instead of developing graphical tools that hide details, the Arch team focuses on documenting everything with clarity.

1.7.3 Why Is KISS Important for the User?

- **Full Control:** You know exactly what's in your system.
- **Easy Maintenance:** Any issue can be easily traced because everything is clear and direct.
- **Flexibility:** The system becomes like a toolbox where you build only what you need.
- **Learning:** KISS makes Arch an excellent educational platform for understanding Linux.

1.7.4 The Difference Between KISS and False Simplification

Some distributions try to be "easy" by building graphical layers that hide complexity. This leads to **"false simplification,"** where the user loses the ability to fully control the system, and when an error occurs, it becomes difficult to fix.

Arch, on the other hand, follows KISS: it keeps the system simple but transparent.

1.7.5 Practical Examples of KISS in Arch

1. **Arch Installation:** There's no complex "graphical installer." You choose the disks, partitioning, kernel, and desktop environment yourself.

2. **Network Setup:** Instead of a huge graphical tool, you can rely on simple tools like `ip` or `systemd-networkd`.
3. **Package Building:** The process is clear and simple through the `PKGBUILD` file.

1.7.6 Criticisms of the KISS Concept in Arch

While KISS is a badge of strength, some criticize it:

- **Steep Learning Curve:** A beginner might find it very difficult at first.
- **Repetitive Work:** Sometimes you need to manually configure things that could have been automated.
- **Time Commitment:** Customizing Arch takes longer than with "ready-to-go" distributions like Ubuntu.

However, these criticisms are actually part of Arch's philosophy: if you want absolute convenience, Arch might not be for you.

1.7.7 KISS in the Words of Arch's Founder

Arch's founder, Judd Vinet, said in an old interview: "Arch is not for everyone. It's for the user who wants to learn and control, not for someone who wants everything ready-made."

This summarizes the idea: **Arch = Freedom + Simplicity + Responsibility**.

 Summary of Section 1.7:

The **KISS** philosophy is what makes Arch Linux unique among distributions. It's a philosophy of clear simplicity that gives the user flexibility and complete control at the expense of a steeper learning curve. Arch doesn't promise absolute ease, but it promises clarity and transparency.

1.8 The Arch Community and Arch Wiki

You can't talk about Arch Linux without mentioning the community behind it. While Arch started as a small distribution founded by Judd Vinet in 2002, the secret to its continued strength today is the active community that maintains and develops it.

1.8.1 The Power of the Arch Community

- **Arch Linux** is not a corporate project; it's a completely community-driven one.
- Thousands of developers and users contribute daily by:
    - Fixing bugs.
    - Updating packages.

    - Adding documentation.
    - Supporting new users.
- The community operates with full transparency: all discussions are open, and all decisions are available on mailing lists and forums.

1.8.2 Arch Wiki: The Premier Linux Encyclopedia

- The **Arch Wiki** is the project's official documentation.
- It started as simple pages explaining how to install Arch, but today it's considered the largest and most comprehensive documentation source for Linux systems in general, even for users of other distributions.

Why is the Arch Wiki special?

1. **Detail:** Every step is explained clearly.
2. **Continuous Updates:** Any change in packages or the system is quickly reflected in the wiki.
3. **Comprehensiveness:** It's not limited to Arch; it includes information about the system as a whole (kernel, systemd, networking, Xorg, Wayland, etc.).
4. **Free and Open:** It's open to everyone, and any user can contribute.

1.8.3 The Community as a Source of Support

- **Forums (Arch Forums):** A place to exchange questions, problems, and solutions.
- **IRC and Matrix Channels:** Direct communication with developers and experienced users.
- **AUR (Arch User Repository):** A huge repository created by the community, containing hundreds of thousands of packages not found in the official repositories.

1.8.4 The Spirit of Sharing and Learning

- In Arch, the philosophy is not just KISS, but also **"teach others."**
- A new user who benefits from the forum or wiki often returns later to help others.
- This continuous cycle of learning and sharing has made Arch more than just a distribution: it's a school for Linux.

1.8.5 Criticisms of the Community

- The community is sometimes described as strict with beginners.
- Some responses can be harsh if the user hasn't read the documentation first.
- However, the reason is that Arch's philosophy is based on self-reliance and reading before asking for help.

1.8.6 Lessons from the Arch Community

- **Transparency creates sustainability:** There are no secrets in development.

- **Shared knowledge is more powerful than individual expertise.**
- **The wiki is better than any complex graphical tool.**

 Summary of Section 1.8:

The **community** is the beating heart of Arch Linux, and the **Arch Wiki** is its brain. Without the community, Arch wouldn't have remained strong and flexible for over two decades. For any new user, the first thing they should learn is how to read the documentation, how to ask clearly, and how to share their experience with others.

1.9 Arch Linux as a Base for Other Distributions

One of the most important signs of a distribution's success is its ability to become a base for other distributions built on top of it. Thanks to its simplicity, modern packages, and powerful repositories, **Arch Linux** has become the foundation for dozens of derivative distributions.

1.9.1 Manjaro Linux: Arch for Beginners

- Manjaro emerged in 2011 from a German team with the goal of making Arch accessible to beginners.
- While Arch focuses on manual installation and system management with precise steps, Manjaro provides:
    - An easy graphical installer.
    - Ready-made driver configurations (especially for NVIDIA cards).
    - Its own repositories that freeze packages for a short period to test them before release, providing greater stability.
- Manjaro is considered a **"gateway"** to the world of Arch, giving the user an experience close to Arch but with added convenience.

1.9.2 EndeavourOS: The Community Spirit

- The background of EndeavourOS goes back to the Antergos project (2002–2019), which aimed to offer Arch with a graphical installer and a ready experience.
- After Antergos stopped, the EndeavourOS project was born in 2019 as a community initiative.
- It differs from Manjaro in that it tries to be closer to "raw" Arch, but it offers:
    - A graphical installer.
    - Multiple desktop environment options (KDE, XFCE, GNOME, etc.).
    - Strong community support that resembles the spirit of Arch itself.
- It is considered an ideal choice for the user who wants an experience almost identical to Arch, but with an easier start.

1.9.3 Helwan Linux: The Egyptian Identity in the World of Arch

- **Helwan Linux** is a distribution based on Arch, created with the goal of combining the power and simple identity of Arch with a desire to create a distinct local experience.
- **Helwan Linux Features:**
    - It relies directly on Arch's repositories, with the addition of local improvements and experiences.
    - Interfaces and tools that make it easier to deal with Pacman, such as the `hpm` (Helwan Package Manager) tool.
    - A focus on simplicity and a clear visual identity, giving it a character different from just another Arch clone.
- Helwan Linux reflects the idea that Arch is not just a distribution, but an open platform that allows any developer or team to create a new project on top of it.

1.9.4 Lessons from Derivative Distributions

- The success of Arch is measured not just by its direct users, but also by the spread of its "children."
- Manjaro, EndeavourOS, and Helwan Linux are examples of how to adapt Arch to different user segments:
    - The beginner who wants ease.
    - The intermediate user who wants flexibility with a faster start.
    - Local communities who want their own identity.

□ Summary of Section 1.9:

Arch Linux is no longer just a standalone distribution; it has become the core of a wide family of distributions. These distributions prove the power of the philosophy behind Arch and its buildability, to meet the needs of a diverse audience around the world.

1.10 Use Cases

Although Arch Linux is a general-purpose distribution that anyone can install, its nature makes it more suitable for specific categories of users. Here we review the most prominent use cases:

1.10.1 For Developers: Access to the Latest Packages

- **Arch Linux** uses a **Rolling Release** model, which means packages are constantly updated.
- This makes it an ideal environment for developers who need:
    - The latest versions of programming languages (Python, Go, Rust, etc.).
    - Modern development libraries and tools without the need to wait for a "new release" of the distribution.
    - The ability to build a completely custom development environment based on their project's needs.
- **Example:** A developer working on an AI project will benefit from the availability of the latest versions of **TensorFlow** or **PyTorch** in Arch's repositories or the AUR.

1.10.2 For Researchers and Academics: Complete Control Over the Environment

- Researchers often need a fully customizable experimental environment.
- Arch provides:
    - The ability to install a custom kernel or a specific version of libraries.
    - High-precision version control, which helps in reproducing scientific results.
    - The flexibility to use tools from the **AUR** or even build packages manually.
- This is especially important in scientific fields like **High-Performance Computing (HPC)**, physics, and applied mathematics.

1.10.3 For the Average User: Between Curiosity and Challenge

- Arch is not the most suitable distribution for beginners or users looking for a "just works" experience.
- However, it is an excellent choice if the user:
    - Wants to learn how Linux works from the inside out.
    - Prefers complete control over pre-configured settings.
    - Has the time and curiosity to embark on a learning journey through manual installation and system management.
- A user who wants a ready-to-use and fast environment without hassle might find distributions like **Ubuntu**, **Fedora**, or even **Manjaro** more suitable.

 Summary of Section 1.10:

- **Arch Linux** is the perfect playground for developers and researchers who need modernity and flexibility.
- The average user can benefit from the experience as a learning journey, but it might not be the best choice for a daily work environment unless they are passionate about exploring the depths of Linux.

1.11 Conclusion of Chapter One

After this tour of the world of **Arch Linux**, we can conclude that Arch is not just another distribution among dozens of others; it is a comprehensive approach and philosophy.

- Its philosophy is based on **simplicity**, **transparency**, and **complete control**.
- Working with it brings you closer to the Linux kernel and helps you understand its internal structure, rather than just settling for the outer shell that other distributions offer.
- Every step in using Arch is a learning journey: from manual installation to managing packages via `pacman`, and even customizing the desktop to your liking.

You could say that **Arch Linux** is like an open "educational workshop" for the user:

- If you're a developer, you'll find the latest packages and tools at your fingertips.
- If you're a researcher, you'll have the ability to control every small and large detail within your environment.
- And if you're an average user passionate about learning, Arch will give you an opportunity to understand Linux in a depth that no other distribution provides.

Thus, learning Arch becomes an entry point to a deeper understanding of Linux itself and gives you greater confidence in dealing with open-source operating systems.

With the end of this introductory chapter, we have laid the theoretical foundation for understanding **Arch Linux**.

Now, it is time to move on to Chapter Two: **Setting up the Archiso build environment**, where we will begin the practical side and discover how we can prepare the working environment to build a custom distribution starting from Arch.

Chapter 2: Setting Up a Build Environment with Archiso

2.1 Introduction to Archiso

What is Archiso?

Archiso is the official framework developed and maintained by **Arch Linux** for building bootable Live ISO images. These images serve various purposes, including distribution, system maintenance, and testing. Essentially, it's **Arch's** "ISO builder," allowing you to:

- Construct a virtual **Arch Linux** instance identical to the official release.
- Or, go further and create a fully customized distribution with a new identity, such as **Helwan Linux**.

Archiso is built on the principle of simplicity: rather than relying on complex tools or external systems, it consists of a collection of scripts and configuration files that enable you to generate a bootable operating system in ISO format.

Why Use Archiso?

To understand its significance, let's ask: "What defines the existence of a Linux distribution?" The answer lies in a readily available **ISO** file that users can download, burn to a USB drive, and boot from. **Archiso** is the tool that empowers you to create this file yourself.

Key reasons for its use include:

1. **Complete Control:** You dictate the packages, configurations, and desktop environment.
2. **Personal Distribution:** Build your personalized system and share it with others.
3. **Maintenance Purposes:** Create a custom ISO for your team, equipped with specialized networking or security tools.
4. **Stability and Testing:** Test a version of your system in a virtual environment like **VirtualBox** before wider distribution.

It can be said that **Archiso** is the "foundry" from which all **Arch**-based projects emerge.

Difference Between Building a Vanilla Arch Installation and Customizing a Distribution like Helwan Linux

To illustrate further, let's compare two scenarios:

Element	Building a Vanilla Arch Installation	Customizing a Distribution like Helwan Linux
Objective	An exact replica of official Arch	A customized version with a new identity
Installed Packages	Core packages only	Desktop environments + Helwan-specific tools
Visual Identity	Arch logo, default wallpapers	Helwan logo, custom themes
Target Audience	Experienced Arch users	New users + professionals
ISO Size	Moderate (700 – 900 MB)	Variable: Very light (Fluxbox) to relatively heavy (Cinnamon)

This comparison highlights that **Archiso** is not merely a copying tool, but a powerful mechanism for crafting a distinct "user experience."

Prerequisites for Building with Archiso

Before embarking on any build process, ensure the following requirements are met:

1. **Suitable Hardware:**
    - A modern processor (**x86_64** is preferred, with virtualization support if testing in **QEMU**/**VirtualBox**).
    - At least **4GB RAM** (**8GB** is recommended).
    - Storage space: **20GB** or more (the build process requires cache and temporary directories).
    - An **SSD** is highly recommended to accelerate compression and decompression operations.
2. **Arch Linux or a Derivative System:**
    - You must have **Arch Linux** installed, or a distribution based on it (like **Helwan Linux**).
    - Reason: **Archiso** relies on specific libraries and tools from the **Arch** environment.
3. **Good Internet Connection:**
    - The build process involves downloading hundreds of packages from mirrors.
    - A weak internet connection can lead to **signature errors**.
4. **Root Privileges:**
    - The build process requires modifying core files and installing packages.
    - Using **sudo** is preferable to logging in directly as root.

Historical Note

Archiso is not a recent development.

- Its origins were very basic: a simple script for building the official **Arch** ISO.
- Over time, users discovered its potential for creating their own custom versions.
- Today, most **Arch**-based distributions (such as **Manjaro**, **EndeavourOS**, **Artix**, and **Helwan Linux**) trace their beginnings back to **Archiso**.

Practical Example to Illustrate the Concept

Imagine this scenario:

- You are working on a standard **Arch** installation.
- You decide to build a lightweight experimental version called **Helwan Light**.
- You simply copy the configuration files from **Archiso**, modify `packages.x86_64`, and remove heavy desktop environments like **GNOME** or **KDE**.
- Within an hour or two, you'll have a 600MB **ISO** file that boots into a very light **Fluxbox** environment.

This demonstrates how **Archiso** significantly streamlines the process and provides a tangible environment for experimentation.

Conclusion (Introduction to the Chapter)

- **Archiso** is the core engine for building any **Arch**-based distribution.
- It enables you to either rebuild a **Vanilla** version or innovate with a new distribution bearing your unique identity.
- The prerequisites are not overly demanding but require suitable hardware and a prepared **Arch** system.
- The upcoming chapters (2.2 – 2.9) will guide you step-by-step, from installing essential tools to testing your distribution in **VirtualBox**.

2.2 intro to archiso

Building an Arch Linux-based distribution using the **archiso** tool revolves around the `releng/` directory, which is the core of the build process. This directory holds all the components that define the final ISO image, from the root filesystem (`airootfs/`) and required package lists to the boot files for both BIOS and UEFI systems.

This document details the function of each file and folder within `releng/`, explaining how these elements combine to create a ready-to-use or installable live environment. Understanding this structure gives a distribution developer the ability to precisely customize every part, from the embedded packages to the user interface on the first boot.

Installing the archiso Tool

The **archiso** tool is responsible for building the ISO image. It's available in the official Arch Linux `extra` repository and can be installed with the following command:

```
sudo pacman -S archiso
```

 Note: It's recommended to perform the build process within a clean Arch environment (or at least within a chroot or VM) to avoid conflicts with the host system.

Getting the releng Profile

The **archiso** tool comes with pre-configured profiles, including `releng` (the same profile Arch Linux uses to produce its official ISO).

To copy the profile to your working directory:

```
cp -r /usr/share/archiso/configs/releng/ ~/archlive/
```

You now have a local copy of the profile at `~/archlive/` that you can freely modify without affecting the original.

archlive/

├── airootfs

│ ├── etc

│ ├── root

│ └── usr

├── bootstrap_packages.x86_64

├── efiboot

├── grub

├── packages.x86_64

├── pacman.conf

├── profiledef.sh

└── syslinux

├── archiso_head.cfg

├── archiso_pxe-linux.cfg

├── archiso_pxe.cfg

├── archiso_sys-linux.cfg

├── archiso_sys.cfg

├── archiso_tail.cfg

├── splash.png

└── syslinux.cfg

This tree diagram provides an overview of the archiso profile layout. In the following sections, each directory and file will be explained in detail.

releng/

The base directory for the **archiso** profile. Everything within it defines:

- Which packages are installed in the ISO.
- Boot settings (EFI / BIOS).
- User and root customization files.
- Pacman settings.
- Bootloader files (grub/syslinux).

---

airootfs/

This is the most critical part, representing the root filesystem (**rootfs**) that will be built inside the ISO. Any file placed here will appear under `/` after booting the system from the ISO.

- `etc/` → System configuration: Contains configuration files like `shadow`, `passwd`, `hostname`, `issue`, etc. Any changes here are reflected in the live system.
- `root/` → The root user's directory within the live environment. Here you can place customization scripts like `customize_airootfs.sh` or dotfiles (`.bashrc`, `.zshrc`).
- `usr/` → Additional user/binary files. Such as `bin/` and `lib/` additions, or any tools you want to pre-install.

---

bootstrap_packages.x86_64

A list of essential packages needed to build the initial ISO environment (**bootstrap environment**). These packages are not included in the final system but are used during the build process to allow **archiso** to `chroot` and continue the build.

---

efiboot/

Contains the EFI bootloader files (**systemd-boot** or **GRUB for UEFI**). This folder holds the `.efi` files and configuration specific to the UEFI boot entry.

---

grub/

Configuration for the **GRUB** bootloader (if you choose to use it). It contains the boot files for the GRUB menu, background, and configuration templates.

□ packages.x86_64

This is the list of packages that will be available inside the live system after booting. This is the most important file for customizing your ISO. Here, you'll list:

- Core utilities (`bash`, `vim`, `nano`).
- Desktop environments or window managers.
- Network and maintenance tools.

□ pacman.conf

The configuration file for the **pacman** package manager inside the live system. You can use it to specify:

- Official repositories.
- Additional repositories (extra/custom).
- Signature verification and other options.

□ profiledef.sh

The "brain" of the profile, defining its core properties. It specifies:

- The distribution's name (`iso_name`, `iso_label`).
- The version (`iso_version`).
- The kernel to use.
- Boot mode settings (UEFI/BIOS).
- Build and staging directories.
- **archiso** follows this file step-by-step to build the ISO.

□ syslinux/

Contains the **Syslinux** bootloader files, used for BIOS/Legacy booting.

- □ `archiso_head.cfg` → The start of the boot menu configuration (title, timeout).
- □ `archiso_pxe-linux.cfg` & `archiso_pxe.cfg` → Configuration for PXE booting (boot over network).
- □ `archiso_sys-linux.cfg` & `archiso_sys.cfg` → Standard boot configuration from the ISO.
- □ `archiso_tail.cfg` → The end of the boot menu configuration.
- □ `splash.png` → A graphical background for the Syslinux screen.
- □ `syslinux.cfg` → The main file that includes the other configuration files (`head`, `tail`, `sys`, `pxe`) and defines the boot menu.

□ In Summary:

- `airootfs/` = The root filesystem built inside the ISO.
- `packages.x86_64` = The packages that will be in the live system.
- `bootstrap_packages.x86_64` = Packages required for the build process only.
- `pacman.conf` = The package manager configuration.
- `profiledef.sh` = The distribution and version properties.
- `efiboot/` + `grub/` + `syslinux/` = Boot files for each mode.

2.3 Core Configuration Files

This chapter serves as the cornerstone of our journey to **customize an Arch Linux distribution**. Here, we'll dive into the heart of the build process, exploring the files that not only **define the content of the final ISO image** but also dictate **how that image is constructed**. Understanding these four files—`profiledef.sh`, `packages.x86_64`, `bootstrap_packages.x86_64`, and `pacman.conf`—is key to achieving **complete control over your project**, enabling you to build a truly **unique distribution**.

We're not just talking about package lists; these files represent an **integrated "recipe"** that `archiso` follows step-by-step to configure the system. Every line in these files has a specific purpose, and every option translates to a particular behavior during the build process or within the live system after booting.

2.3.1 `profiledef.sh`: The Mastermind Behind Your Custom Distribution

This file is the **core of any `archiso` profile**. It's a **Bash shell script** containing all the definitions and settings that determine your ISO image's identity, its boot behavior, and its build process. Think of it as the **"to-do list"** that the `archiso` tool follows to produce your live system.

Bash

```
#!/usr/bin/env
bash
#
shellcheck disable=SC2034
```

- `#!/usr/bin/env bash`: This is the well-known **"shebang" line**, which instructs the system to execute this script using the Bash interpreter.
- `# shellcheck disable=SC2034`: This is a comment specifically for the `shellcheck` tool (a Bash script analysis tool). Here, we are telling `shellcheck` to ignore a specific warning related to unused variables (SC2034), as some variables might be intended for specific purposes or used internally by `archiso`.

ISO Metadata

These variables define the **identity and attributes of the final ISO image**.

Bash

```
iso_name="Helwan-Linux-stable"
iso_label="Helwan-Linux-stable-v1.1"
iso_publisher="helwanlinux <helwanlinux@gmail.com>"
iso_application="Helwan Linux Live/Rescue DVD"
iso_version="v1.1"
```

- `iso_name`: The **name of the ISO file** that will be generated. In our example, it will be `Helwan-Linux-stable-x86_64.iso` (typically, `archiso` automatically appends the processor architecture).
- `iso_label`: The **ISO Label**. This is the text string that appears when the ISO is mounted or booted from (e.g., in a boot manager menu). It should be concise (usually a maximum of 32 characters, but 16 characters is best for wider compatibility).
- `iso_publisher`: **Publisher information**. This includes the name of the entity responsible for the distribution and a contact point (an email address in this example).
- `iso_application`: A **brief description of the ISO's purpose**. It clarifies its intended use, such as "Live/Rescue DVD" or "System Installer."
- `iso_version`: The **version number of the distribution**. This is important for tracking updates and identifying specific releases.

Core Build Settings

These variables define the **structure and essential components of the build process**.

Bash

```
install_dir="arch"
buildmodes=('iso')
arch="x86_64"
pacman_conf="pacman.conf"
airootfs_image_type="squashfs"
```

- `install_dir`: The **installation directory within the ISO**. When a user mounts the ISO, the system files will be placed in this path. The default for Arch Linux is `arch`.
- `buildmodes`: The **available build modes**.
    - `iso`: Indicates that we will be building a bootable ISO image.
    - Other modes might include `tar` (to create a filesystem archive) or `vendor` (to create custom images for specific hardware), but `iso` is the most common.
- `arch`: The **target processor architecture**. Here, `x86_64` signifies that the image is designed for 64-bit systems.
- `pacman_conf`: The **name of the Pacman configuration file**. This points to the `pacman.conf` file located in the same profile directory, which will be used to configure the package manager within the live system.
- `airootfs_image_type`: The **compression type for the root filesystem**.
    - `squashfs`: This is the standard type for Live Systems, offering excellent compression and read speed.
    - Other options might include `ext4`, but `squashfs` is the most commonly used for ISO images.

Compression Options and Tools

These variables specify **how the root filesystem (`airootfs`) and other files are compressed**.

Bash

```
airootfs_image_tool_options=('-comp' 'xz' '-Xbcj' 'x86' '-b' '1M' '-Xdict-
size' '1M')
bootstrap_tarball_compression=('zstd' '-c' '-T0' '--auto-threads=logical' '--
long' '-19')
```

- `airootfs_image_tool_options`: Options for the `squashfs` compression tool.
  - `'-comp' 'xz'`: Uses the `xz` compression algorithm, which provides a very high compression ratio at the cost of longer processing time.
  - `'-Xbcj' 'x86'`: Specifies a special filter (x86 Code Branch) to apply compression to x86 architecture code, improving the compression ratio.
  - `'-b' '1M'`: The block size that the tool will use. A larger size generally means better compression but higher memory usage during compression.
  - `'-Xdict-size' '1M'`: The compression dictionary size. A larger dictionary generally leads to better compression.
- `bootstrap_tarball_compression`: Compression options for the archive used to build the initial environment.
  - `('zstd' '-c' '-T0' '--auto-threads=logical' '--long' '-19')`: Here, we use `zstd`, a very fast and efficient compressor.
    - `'-c'`: Writes output to standard output (stdout).
    - `'-T0'`: Uses all available processor cores (to maximize speed).
    - `'--auto-threads=logical'`: Automatically adjusts the number of threads based on the number of logical cores.
    - `'--long'`: Enables the "long" mode, which increases compression effectiveness (beneficial for large files).
    - `'-19'`: The compression level (19 is the highest, meaning maximum compression at the expense of time).

Boot Modes

This variable is one of the most critical, as it **determines how the ISO can be booted**.

Bash

```
bootmodes=('bios.syslinux.mbr'
           'bios.syslinux.eltorito'
           'uefi-ia32.systemd-boot.esp' 'uefi-x64.systemd-boot.esp'
           'uefi-ia32.systemd-boot.eltorito' 'uefi-x64.systemd-
boot.eltorito')
```

This reflects the sophistication and compatibility of `archiso`. Let's break down these strings:

- `bios.syslinux.mbr`:
  - `bios`: Refers to the **BIOS (Legacy) boot mode**.
  - `syslinux`: Uses **Syslinux as the bootloader**.
  - `mbr`: Employs the **Master Boot Record (MBR)** to create an executable boot sector on the disk, allowing booting from older hardware.
- `bios.syslinux.eltorito`:
  - `bios`: **BIOS mode**.

    - `syslinux`: **Syslinux bootloader**.
    - `eltorito`: Uses the **El Torito standard** to create a bootable CD/DVD. This is the standard format for bootable ISO images.
- `uefi-ia32.systemd-boot.esp`:
    - `uefi`: Refers to the **UEFI (Unified Extensible Firmware Interface) boot mode**.
    - `ia32`: Indicates support for **32-bit (x86) processors**.
    - `systemd-boot`: Uses **systemd-boot** (formerly gummiboot) as the UEFI bootloader.
    - `esp`: Refers to the structure of the files within the ISO that **simulates an EFI System Partition (ESP)**.
- `uefi-x64.systemd-boot.esp`:
    - `uefi`: **UEFI mode**.
    - `x64`: Supports **64-bit (x86_64) processors**.
    - `systemd-boot`: The **systemd-boot bootloader**.
    - `esp`: **ESP file structure**.
- `uefi-ia32.systemd-boot.eltorito`:
    - `uefi`: **UEFI mode**.
    - `ia32`: Supports **32-bit**.
    - `systemd-boot`: The **systemd-boot bootloader**.
    - `eltorito`: Uses the **El Torito standard** to build a bootable UEFI ISO image.
- `uefi-x64.systemd-boot.eltorito`:
    - `uefi`: **UEFI mode**.
    - `x64`: Supports **64-bit**.
    - `systemd-boot`: The **systemd-boot bootloader**.
    - `eltorito`: The **El Torito standard** for creating a bootable UEFI ISO.

Practical Application: This means your resulting image will be capable of booting from most modern (UEFI) and older (BIOS) hardware, using the default Arch Linux bootloaders (`systemd-boot` for UEFI and Syslinux for BIOS).

File Permissions

This section defines the **permissions for critical files and directories within the root filesystem (`airootfs`)** to ensure security and proper operation.

Bash

```
file_permissions=(
  ["/etc/shadow"]="0:0:400"
  ["/root"]="0:0:750"
  ["/root/.automated_script.sh"]="0:0:755"
  ["/root/.gnupg"]="0:0:700"
  ["/usr/local/bin/choose-mirror"]="0:0:755"
  ["/usr/local/bin/Installation_guide"]="0:0:755"
  ["/usr/local/bin/livecd-sound"]="0:0:755"
  ["/etc/polkit-1/rules.d"]="0:0:750"
  ["/etc/sudoers.d"]="0:0:750"
)
```

Here, the format `["file_path"]="owner:group:permissions"` is used.

- `owner:group`: The **User ID (UID) and Group ID (GID)**. `0:0` typically means the `root` user and the `root` group.
- `permissions`: Permissions in **octal notation**.
    - `400`: Read-only for the owner (`r--------`).
    - `750`: Read, write, and execute for the owner (`rwxr-x---`); read and execute for the group.
    - `755`: Read, write, and execute for the owner (`rwxr-xr-x`); read and execute for others.
    - `700`: Execute only for the owner (`rwx------`).

Importance of These Permissions:

- `/etc/shadow`: Contains **encrypted passwords**. It should be owned by `root` and only readable by `root` (`400`).
- `/root`: The **root user's directory**. `750` means `root` can read, write, and execute, while the group can only read and execute (useful if specific groups are created).
- `/root/.automated_script.sh`: If you have **automated scripts**, they should be executable (`755`).
- `/root/.gnupg`: For directories containing GPG keys, it's best to **restrict access to `root` only** (`700`).
- `/usr/local/bin/`: This path is the standard location for adding **custom scripts**. Setting `755` permissions makes them executable by everyone.
- `/etc/polkit-1/rules.d` and `/etc/sudoers.d`: These directories contain **privilege escalation rules**. Restricting access to them (`750`) prevents unauthorized users from modifying them.

Chapter Summary

The `profiledef.sh` file is a **treasure trove for controlling the ISO build process**. We've seen how this file defines:

- The **distribution's identity** (name, version, publisher).
- The **system's architecture** (processor, install directory, system image type).
- **Boot features** (BIOS, UEFI, Syslinux, systemd-boot).
- **Security settings** (file permissions).
- **Compression tools** for optimizing ISO size and build efficiency.

Your deep understanding of this file will empower you to customize your distribution beyond just a package list, extending to the initial system behavior. In subsequent chapters, we'll expand on these definitions as we delve into the `airootfs/` directories and boot files.

2.3.2 `packages.x86_64`: The Live System Package List

If `profiledef.sh` is the blueprint, then `packages.x86_64` is the actual list of system components. This file is a straightforward text-based list of packages that `archiso` will install within the **Live System**.

Selection Philosophy

When building a custom distribution, carefully consider the packages you include. There's a balance between providing a feature-rich environment and keeping the **ISO image size** lean.

Example Content

Plaintext

```
# Core System
base
base-devel
linux
linux-firmware
nano
vim

# Desktop Environment (Example: GNOME)
xorg-server
gnome
gnome-extra

# Networking and Connectivity
networkmanager
dhcpcd
wpa_supplicant

# System Utilities
htop
neofetch
pacman-contrib
```

Tips for Customizing the List:

- **Start with `base` and `linux`**: These packages are essential for providing a bootable system.
- **Add Essential Utilities**: Tools like `nano`, `vim`, and `git` are often necessary for users.
- **Define Your Desktop Environment**: Choose a desktop environment (e.g., GNOME, KDE Plasma, XFCE) or a window manager (e.g., i3, Sway) along with all its necessary dependencies.
- **Consider Hardware**: If your distribution targets specific hardware, think about including proprietary drivers (e.g., for Wi-Fi or graphics cards).

- **Package Management**: `archiso` uses `pacman`, but you can add helper tools like `yay` if desired.

2.3.3 bootstrap_packages.x86_64: Building Process Essentials

Unlike `packages.x86_64`, the packages listed in `bootstrap_packages.x86_64` are not part of the final live system. Instead, they are the packages that `archiso` itself requires to create an isolated build environment (chroot) and install other packages. These packages are crucial for the efficient execution of the assembly process.

Example Content of `bootstrap_packages.x86_64` (e.g., "Helwan Linux"):

```
arch-install-scripts
archiso
base
base-devel
linux
linux-firmware
```

Explanation of these Packages:

- **`arch-install-scripts`**: This package provides essential tools for installing Arch Linux, most notably the `arch-chroot` utility. This utility allows `archiso` to enter the environment of the system being built to perform installations and modifications.
- **`archiso`**: This is the tool itself! It must be available within the build environment.
- **`base`**: Provides the minimal set of packages necessary for an Arch Linux system to function, including `systemd` and other fundamental utilities.
- **`base-devel`**: Includes essential software development tools (such as `gcc`, `make`) that `archiso` might require for building specific components (though less common in most ISO build processes).
- **`linux`** **and** **`linux-firmware`**: The kernel and its associated firmware, which are necessary for constructing a basic operating environment.

When to Modify This File:

In most scenarios, you won't need to modify `bootstrap_packages.x86_64`. The default list is sufficient for building most Arch Linux distributions. You would only need to edit it if you are undertaking a highly specialized build or encountering specific issues within the build environment itself.

2.3.4 `pacman.conf`: The Control Center for Your Custom World

The `pacman.conf` file is the central control hub for `pacman`, the package manager that's the backbone of software management in Arch Linux. When you're building an ISO image using `archiso`, this file dictates how packages are fetched and installed, both within the live system environment and during the initial build phase (bootstrap).

Let's break down this file line by line, emphasizing the options that matter most to you as a distribution developer:

`[options]` Section (General Settings)

This section acts as `pacman`'s comprehensive dashboard.

Paths:

```
#RootDir = /
#DBPath = /var/lib/pacman/
#CacheDir = /var/cache/pacman/pkg/
#LogFile = /var/log/pacman.log
#GPGDir = /etc/pacman.d/gnupg/
#HookDir = /etc/pacman.d/hooks/
```

These paths specify where `pacman` stores its databases (`DBPath`), where it keeps downloaded packages (`CacheDir`), where it writes operation logs (`LogFile`), and where it looks for GPG keys (`GPGDir`) and auxiliary tools (`HookDir`).

- **In the context of `archiso`:** When `pacman` runs within a `chroot` environment during the build process, these paths typically point to temporary locations within the build directory. For instance, `CacheDir` might reside within a temporary folder to prevent polluting the host system's cache.
- **For customization:** You'll rarely need to alter these paths when building an ISO, but understanding them is key to grasping where `pacman` operates internally.

`HoldPkg` and `IgnorePkg` / `IgnoreGroup`:

Ini, TOML

```
HoldPkg = pacman glibc
IgnorePkg =
IgnoreGroup =
```

- **`HoldPkg`:** This prevents specified packages from being automatically upgraded. This is crucial for maintaining system stability, especially for core packages like `pacman` itself and `glibc`.
- **`IgnorePkg`:** A list of packages that you **never** want `pacman` to update.

- **`IgnoreGroup`:** A list of package groups that you don't want to update.
- **In the context of `archiso`:** Keeping `pacman` and `glibc` in `HoldPkg` is a good practice to ensure the build process isn't broken by unexpected changes in these vital packages.

`Architecture = auto`:

This option defines the target architecture. `auto` means `pacman` will attempt to detect the architecture automatically (e.g., `x86_64`).

- **In the context of `archiso`:** Since you're building an ISO for a specific architecture, you could explicitly set it here (e.g., `Architecture = x86_64`), but `auto` generally works well.

`ParallelDownloads = 5`:

This option specifies how many packages `pacman` can download concurrently. Increasing this number (e.g., to 5 or 10) can significantly speed up the download process, especially with a good internet connection.

- **In the context of `archiso`:** Faster downloads translate to a faster ISO build, making this a very beneficial option.

`SigLevel` and `LocalFileSigLevel`:

```
SigLevel = Required DatabaseOptional
LocalFileSigLevel = Optional
```

These settings define the level of verification for package digital signatures.

- **`SigLevel = Required DatabaseOptional`:** This means `pacman` will demand a valid signature (`Required`) for packages downloaded from remote repositories, but it will accept packages with optional signatures (`DatabaseOptional`) or even unsigned packages from repository databases.
- **`LocalFileSigLevel = Optional`:** This indicates that packages installed from local files (e.g., manually copied ones) can be unsigned or optionally signed.
- **In the context of `archiso`:** This setting ensures a certain level of security when fetching packages while remaining flexible enough not to disrupt the build if there's a temporary signature issue.

`[repositories]` Section (Package Sources)

This is where you define where `pacman` will search for packages. The order here is critical, as `pacman` will use the **first repository** that contains the requested package.

Official Repositories:

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

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

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

- **`[core]`, `[extra]`:** These are the fundamental repositories in Arch Linux.
- **`Include = /etc/pacman.d/mirrorlist`:** This tells `pacman` to use the `/etc/pacman.d/mirrorlist` file to select the best mirror for `core` and `extra`.
- **`[multilib]`:** This is disabled by default. If you need support for 32-bit applications on a 64-bit system, you'll need to uncomment it (remove the `#`).
- **In the context of `archiso`:** These settings enable `archiso` to access the official Arch Linux repositories to download all the packages you've specified in `packages.x86_64` and `bootstrap_packages.x86_64`.

Your Custom Repository (Example: `helwan`)

```
 [helwan]
SigLevel = Optional TrustedOnly
Server = https://helwan-linux.github.io/$repo/$arch
```

This section is a live example of how to add your own custom repository.

- **`[helwan]`:** The name of the repository (must be unique).
- **`SigLevel = Optional TrustedOnly`:** Here, you're telling `pacman` that it prefers packages to be signed (`Optional`), but it will trust them if they come from a trusted repository (`TrustedOnly`) even if explicit signatures aren't present. This can be useful for custom repositories where not every package might be consistently signed.
- **`Server = https://helwan-linux.github.io/$repo/$arch`:** This is the URL of the repository.
    - `$repo` will be automatically replaced with the repository name (i.e., `helwan`).
    - `$arch` will be replaced with the target architecture (e.g., `x86_64`).

Why it matters: This allows you to include your own custom packages, or even modified versions of Arch packages, directly into your ISO image. You'll need a web server (or even a local repository) to host these packages.

Local Custom Repository (Example: `custom`)

```
#[custom]
#SigLevel = Optional TrustAll
#Server = file:///home/custompkgs
```

This is another example of a custom repository, but one that relies on the local filesystem.

- **`SigLevel = Optional TrustAll`:** Here, `TrustAll` means `pacman` will not verify any signatures at all. This is not recommended for public repositories due to security concerns but can be useful for quick experimentation or in isolated environments.
- **`Server = file:///home/custompkgs`:** Points to a folder path on the local filesystem containing the packages.
- **In the context of `archiso`:** If you're storing your custom packages locally within the profile's files, you can use this type of setup.

Practical Illustrative Example

Let's say you're building a "Helwan Linux" distribution and want to include a custom package named `helwan-tools` that isn't available in the official Arch repositories.

1. **Create Your Custom Repository:**
    - Create a directory for your repository, e.g., `~/my_helwan_repo`.
    - Place your `helwan-tools.pkg.tar.zst` (or another format) package file inside this directory.
    - Build the repository database using `repo-add`:

      Bash

      ```
      repo-add ~/my_helwan_repo/helwan.db.tar.gz
      ~/my_helwan_repo/helwan-tools-*.pkg.tar.zst
      ```

    - Upload the contents of `~/my_helwan_repo` (including `helwan.db.tar.gz` and `helwan-tools-*.pkg.tar.zst`) to a web server, or use `file://` if it's local.
2. **Modify `pacman.conf`:** If you uploaded it to `https://helwan-linux.github.io/helwan/x86_64/`, your `pacman.conf` section would look like the example provided:

   ```
    [helwan]
   SigLevel = Optional TrustedOnly
   Server = https://helwan-linux.github.io/$repo/$arch
   ```

3. **Modify `packages.x86_64`:** Add your custom package to the list:

   Plaintext

   ```
   # ... other packages ...
   helwan-tools
   # ... additional packages ...
   ```

Now, when `archiso` builds the ISO, `pacman` will know where to fetch `helwan-tools` from and will install it into the live system.

`pacman.conf` from `profiledef.sh` (Additional Explanation)

You also provided an example snippet from a `profiledef.sh` file illustrating the use of `pacman.conf` as an external file:

Bash

```
pacman_conf="pacman.conf" # Refers to the pacman.conf file located in the
same profile directory
```

This line in `profiledef.sh` signifies that `archiso` will use the file named `pacman.conf`, located within the same profile directory (e.g., `releng/` or `~/archlive/` in your example), as the configuration file for `pacman`. This is the default and recommended behavior, as it keeps all `pacman` settings consolidated with your specific profile.

Chapter Summary

The `pacman.conf` file is a potent tool for customizing your package sources and controlling how packages are fetched and installed. By managing it carefully, you can ensure your live system has access to all the packages it needs, including custom ones, in a secure and efficient manner. Understanding these settings empowers you with complete control over the "repositories" your distribution relies upon.

.

Summary of Chapter

We've covered significant ground in understanding the core components of an `archiso` profile. You can now visualize how these files integrate:

- `profiledef.sh` is read to define everything about the image.
- `bootstrap_packages.x86_64` is used to build an initial `chroot` environment.
- Within the `chroot`, `pacman.conf` configures `pacman`.
- `packages.x86_64` is followed to install all required packages.
- The image build is completed based on the remaining instructions.

In the next chapter, we'll delve deeper into the `airootfs/` directory, which is where you'll apply your final touches and unique customizations.

2.4.1 `airootfs/` Directory

After understanding the core files, the reader moves on to how to customize the filesystem itself. This chapter explains how to add your own files or directly modify system settings.

- **Explanation** of the role of the `airootfs/` directory as the root filesystem environment.
- **Explanation** of subdirectories:
    - `etc/`: System configurations.
    - `root/`: The root user's directory.
    - `usr/`: Additional binary files.

2.4.1.1 A Look into the `/etc/` Directory

The `/etc/` directory contains configuration files and settings for most programs and the system itself. It's where you customize the behavior of the system and its services.

 X11 * `xorg.conf.d/30-touchpad.conf`: Configures touchpad settings for the Xorg graphical environment, such as touch sensitivity and scrolling.

 default * `grub`: Configuration file for GRUB, the bootloader, which controls how the system starts and the available boot options.

 group * This file defines the groups present on the system and their members.

 gshadow * Contains encrypted group passwords (if used).

 hostname * Contains the hostname for your machine.

 lightdm * `lightdm.conf`: Main settings for LightDM, a display manager for the graphical user interface. * `slick-greeter.conf`: Greeter (login screen) settings for LightDM.

 locale.conf * Defines the system's language and region settings, such as date, time, and number formatting.

 localtime * Indicates the configured local time for the system.

 mkinitcpio.conf * Configuration file for `mkinitcpio`, used to create `initramfs` images (an initial filesystem loaded before the main filesystem).

 mkinitcpio.conf.d * `archiso.conf`: A configuration file specific to Archiso, a tool for creating bootable Arch Linux ISO images.

 mkinitcpio.d * `linux.preset`: Specifies which kernel modules should be included in the `initramfs` image.

 modprobe.d * `broadcom-wl.conf`: Defines load options for the Broadcom wireless driver kernel module.

 motd * Message Of The Day: Displayed to users upon login.

 pacman.conf * The primary configuration file for `Pacman`, Arch Linux's package manager, defining repositories and update settings.

 pacman.d * `hooks/uncomment-mirrors.hook`: A `Pacman` hook that uncommentsthe mirror options for update settings. * `hooks/zzzz99-remove-custom-hooks-from-airootfs.hook`: Another `Pacman` hook designed to remove custom hooks from `airootfs`.

 passwd * Contains essential user information, such as usernames, UIDs, and home directories.

 polkit-1 * `rules.d/49-nopasswd_global.rules`: Rules for `PolicyKit`, allowing certain actions without requiring a password.

 resolv.conf * Specifies the DNS servers the system uses to resolve domain names.

 shadow * Contains encrypted user passwords and other authentication data.

 skel * The "skeleton" directory: Contains default files and settings that are copied to a new user's home directory upon creation. * `.Xresources`, `.bashrc`: Common configuration files for the X environment (e.g., appearance settings) and the Bash shell (commands and aliases). * `.cinnamon`: Specific settings and configurations for the Cinnamon desktop environment, including settings for applets and widgets. * `.config/dconf/user`: DConf system configuration data. * `.config/nemo/desktop-metadata`: Metadata for the desktop in the Nemo file manager. * `.icons/default/index.theme`: Defines the default system icons. * `Templates`: Templates for new files (e.g., spreadsheets, documents, scripts).

 ssh * `sshd_config.d/10-archiso.conf`: Additional SSH server configuration settings specific to the Archiso build.

 sudoers.d * `g_wheel`: Configuration file for `sudo`, granting members of the `wheel` group privileges to execute commands as the root user.

 systemd * Configuration directory for `systemd`, the init system in many modern Linux distributions. * `journald.conf.d/volatile-storage.conf`: Settings for temporary storage of `journald` logs. * `logind.conf.d/do-not-suspend.conf`: Prevents the system from entering suspend mode. * `network`: `systemd` network configuration, defining how network interfaces (Ethernet, WLAN, WWAN) are configured. * `network.conf.d/ipv6-privacy-extensions.conf`: Enables privacy extensions for IPv6. * `resolved.conf.d/archiso.conf`: Specific settings for `systemd-resolved` (name resolution system) for Archiso. * `system`:

Contains various `.service` and `.target` files that define how services run and the system boots. Examples include `display-manager.service` (to start the graphical interface), `NetworkManager.service` (for network management), `sshd.service` (for the SSH server), and `reflector.service` (to optimize `Pacman` mirrors). * `system-generators`: System generators, such as `systemd-gpt-auto-generator`, which aids in automatically configuring GPT partitions. * `zram-generator.conf`: Configuration to create ZRAM (memory compression) partitions for improved system performance.

 xdg * `reflector/reflector.conf`: Configuration file for `reflector`, a tool that helps find and update the fastest `Pacman` mirrors.

2.4.1.2 `/root` Directory Contents

The `/root` directory serves as the **Home Directory** for the **root user**, which possesses complete administrative privileges on a Linux system. Any modifications or files placed here are exclusive to the root user and typically do not affect other users.

 `root/`

- **`.automated_script.sh`**:
  - **Description**: This file is a **shell script** likely intended for executing automated or **scripted tasks**. The leading dot (`.`) indicates it is a **hidden file**.
  - **Technology**: Scripts of this nature are used to run a series of commands automatically, saving time and effort for the root user when performing repetitive actions. It could include commands for installing software, configuring the system, performing backups, or other administrative operations.
  - **Usage**: This script can be executed manually by the root user, or it may be invoked by another process or a specific system event.
- **`.gnupg/`**:
  - **Description**: This directory contains configuration and data for **GnuPG** (GNU Privacy Guard). GnuPG is a tool for encrypting and decrypting data and digital signing. The leading dot (`.`) makes the directory **hidden**.
  - **Technology**:
    - `scdaemon.conf`: This is a **configuration file** for `scdaemon` (Smartcard Daemon), a component of GnuPG that handles smart cards and other cryptographic hardware. This file defines how `scdaemon` should interact with these devices, such as specifying particular paths or connection options.
  - **Usage**: GnuPG is used to secure communications, verify data integrity, and manage keys. The presence of this directory in `/root` suggests that the root user may be setting up and performing encryption or signing operations that require advanced key management.
- **`.zlogin`**:
  - **Description**: This file is a **script executed automatically** upon the root user's login to the system (particularly upon logging in via the terminal/shell). It is analogous to `.bash_profile` or `.profile` but is specific to `zsh` (Z shell) or other shells that support it, commonly found in certain environments. The leading dot (`.`) makes it a **hidden file**.
  - **Technology**: When a user logs in, the shell reads this file to execute the commands contained within. It can be used to set environment variables, launch specific programs upon session startup, display custom welcome messages, or load user-specific configurations.
  - **Usage**: This allows the root user to customize their working environment with each login, ensuring certain tools or settings are ready immediately upon session commencement.

In summary, within this context, the `/root` directory contains **user-specific utilities and scripts** for the root user, either for automating tasks (`.automated_script.sh`), managing encryption and security (`.gnupg/scdaemon.conf`), or personalizing the login environment (`.zlogin`).

usr/
 usr/
├── local
│ ├── bin
│ │ ├── Installation_guide
│ │ ├── choose-mirror
│ │ ├── launch
│ │ └── livecd-sound
│ └── share
│ ├── livecd-sound
│ └── asound.conf.in
└── share
├── backgrounds
│ ├── bg.png
│ ├── helwan
├── grub
│ └── themes
│ └── helwan
└── nemo
└── actions

└── count_lines.nemo_action

2.4.1.3 Understanding the File System Structure - The `/usr` Directory

After grasping the core concepts of essential files and the role of the `airootfs` directory as the root file system environment, this chapter moves on to explore one of the most significant directories in any Linux system: the `/usr` directory. This directory serves as the main repository for applications and shared data that are not critical for the initial system bootstrap process but are vital for running the user environment and additional functionalities.

The Role of the `/usr` Directory

The name `/usr` is a traditional abbreviation for "Unix System Resources" or "User System Resources." Although this abbreviation dates back to the early days of Unix systems, its current function is to contain executables, libraries, header files, and documentation that form the bulk of the user's environment.

A key feature of the `/usr` directory is that its contents are read-only in most cases. This means the system can function even if the `/usr` partition is mounted as a read-only file system. This design ensures system stability and prevents accidental changes to core programs.

Exploring the Main Subdirectories within `/usr`

The `/usr` directory is divided into several primary subdirectories, each serving a specific purpose:

- `/usr/bin`: Contains the fundamental executables and programs that all users can run. Unlike the essential bootstrap commands in `/bin`, this directory holds most of the commonly used applications and tools.
- `/usr/sbin`: Contains executables and programs intended for the administrator (`root`), such as system and network administration tools.
- `/usr/lib`: Contains static and shared libraries required by the executables in `/usr/bin` and `/usr/sbin`.
- `/usr/local`: This is the designated location for programs that are compiled from source or manually installed by the administrator. This prevents conflicts with system-provided packages (located in `/usr/bin`, `/usr/lib`, etc.).
- `/usr/share`: Contains non-executable data files shared by programs, such as general configuration files, icons, backgrounds, man pages, and other documentation.

A Detailed Look at the `/usr` Contents in the Archiso Context

In the context of Archiso, the tool used to create bootable Arch Linux ISO images, the `/usr` directory within `airootfs` reflects the standard Arch Linux structure, with some modifications or additions specific to the live environment or installer tools. Let’s take a detailed look at the specific structure you provided:

`/usr/local`

This directory is the standard location for installing programs and applications that are not part of the original distribution but were manually installed by the user or administrator, often after being compiled from source. This ensures separation between core system packages managed by a package manager (like Pacman) and custom applications, reducing the risk of dependency conflicts or overwriting system files.

`/usr/local/bin` This directory contains the executables installed in `/usr/local`.

- **`Installation_guide`**: This could be a text file or a script containing detailed instructions on how to install programs or customize the system, either in the live environment or after installation. It might include steps, tips, and best practices.
- **`choose-mirror`**: This script likely helps in selecting the nearest or fastest mirror for the Arch Linux package repositories, which is crucial for accelerating package download and update operations.
- **`launch`**: A generic name for a script or program used to start a specific application or service. This could be designed to launch a desktop environment, a specific tool, or an initialization process.
- **`livecd-sound`**: This refers to a script or program related to sound management or configuration in the Live CD/USB environment. It may be responsible for loading audio drivers, setting initial sound levels, or configuring audio devices.

`/usr/local/share` This directory contains non-executable files associated with programs installed in `/usr/local`.

- **`livecd-sound`**: This subdirectory might contain configuration files or additional data related to handling sound in the live environment.
- **`asound.conf.in`**: This is a configuration file for ALSA (Advanced Linux Sound Architecture). The `.in` suffix often indicates it's a template file that will be processed or modified (perhaps by a script) to create the final `asound.conf` configuration file used by the system. This file defines how sound devices, mixing, and other audio components operate.

`/usr/share`

This directory is a large repository of shared, non-executable files that are necessary for programs to function or to provide a visual interface to the user.

`/usr/share/backgrounds` This directory contains desktop background images that users can choose from.

- **`bg.png`**: This is a default or commonly used background image.
- **`helwan`**: This is a subdirectory, and it likely contains a collection of wallpapers with a specific theme or design related to "Helwan" (a name that could refer to a particular project, logo, or culture).

`/usr/share/grub` This directory contains supporting files for the GRUB (Grand Unified Bootloader) boot manager.

- **`themes`**: A directory that contains theme files for GRUB, allowing for the customization of the boot screen.
- **`helwan`**: A subdirectory representing a custom theme for GRUB, which may include special logos, images, and layouts for this theme.

`/usr/share/nemo` This directory contains data specific to the Nemo file manager, which is often used in desktop environments like Cinnamon.

- **`actions`**: This directory is dedicated to custom Nemo Actions. These actions allow users to add new options to the right-click context menu in Nemo, facilitating the execution of specific tasks on files.
- **`count_lines.nemo_action`**: This file defines a custom action for Nemo that counts the lines in selected files. When a user right-clicks on a file (or group of files) and selects this action, it will execute a script or command that counts the lines in those files, possibly displaying the result in a terminal window or a message box. This is an excellent example of how to extend the functionality of a file manager.

Conclusion

The `/usr` directory represents the software and application heart of a Linux system, hosting the vast majority of programs, tools, and libraries used by both users and administrators. In the context of Archiso, this directory mirrors the standard Arch Linux structure, while also highlighting some files and scripts that support the live environment and customization tools. Understanding the organization of this directory is crucial for comprehending how the system works, how to customize it, and how to manage applications and services effectively.

2.5 Grub

The GRUB (Grand Unified Bootloader) configuration files, such as `grub.cfg` and `loopback.cfg`, are the cornerstone of the boot process for Unix-like operating systems. The `grub.cfg` file defines the boot menu and its options, while `loopback.cfg` specializes in booting systems directly from ISO images.

This chapter provides a detailed explanation of these two critical files, clarifying the function of each command and analyzing its role in the boot process. By understanding how to load modules, configure graphics mode, and recognize file systems, users can gain full control over and customize their boot experience, whether for debugging, system administration, or installing a new operating system.

Through this professional and technical explanation, you will acquire a deep understanding of the internal components of the boot process, enabling you to handle GRUB environments with flexibility and confidence.

grub/
 grub/
├── grub.cfg

└── loopback.cfg

Detailed Explanation of the `grub.cfg` File

`grub.cfg` file is the main configuration file for **GRUB (Grand Unified Bootloader)**, a very common boot loader in Unix-like operating systems. This file determines how the boot menu is displayed, what boot options are available, and how each option is booted

1. Loading GRUB Modules

Code Snippet:

```
insmod part_gpt
insmod part_msdos
insmod fat
insmod iso9660
insmod ntfs
insmod ntfscomp
```

```
insmod exfat
insmod udf
```

- `insmod module_name`: This command loads a specific **module** into GRUB. Modules are loadable components that extend GRUB's core functionality.
- `part_gpt` and `part_msdos`: These commands load the necessary modules to recognize **disk partition tables** of types **GPT (GUID Partition Table)** and **MBR (Master Boot Record)**. This allows GRUB to understand how hard disks are partitioned.
- `fat`, `iso9660`, `ntfs`, `ntfscomp`, `exfat`, `udf`: These modules are essential for recognizing different **file systems**.
    - `fat`: The FAT16/FAT32 file system, common on USB drives and older storage devices.
    - `iso9660`: The standard file system used on CD/DVD discs.
    - `ntfs` and `ntfscomp`: The Windows file system (NTFS) with compression support.
    - `exfat`: A file system common for large disks and removable storage.
    - `udf`: The file system often used on DVD and Blu-ray discs.

---

2. Graphics Mode Settings

Code Snippet:

```
# Use graphics-mode output
insmod all_video
insmod font
if loadfont "${prefix}/fonts/unicode.pf2" ; then
    set gfxmode="auto"
    terminal_input console
    terminal_output console
fi
```

- `insmod all_video`: This command loads a module that supports video output for all compatible devices.
- `insmod font`: This command loads the font module.
- `if loadfont "${prefix}/fonts/unicode.pf2" ; then ... fi`: This checks if the `unicode.pf2` font file (which supports a wide range of characters) can be loaded. If the load is successful:
    - `set gfxmode="auto"`: GRUB is configured to automatically use the best available graphics mode.
    - `terminal_input console` and `terminal_output console`: This directs GRUB's input and output to the console, meaning the graphics will appear directly on the system's screen.

---

3. Enabling Serial Console

Code Snippet:

```
# Enable serial console
if serial --unit=0 --speed=115200; then
    terminal_input --append serial
    terminal_output --append serial
fi
```

- `serial --unit=0 --speed=115200`: This command initializes the serial port for unit 0 (usually COM1) at a speed of 115,200 bits per second.
- `terminal_input --append serial` and `terminal_output --append serial`: This adds the serial port as a destination for both input and output. This is useful for debugging or remote system administration via a serial connection.

---

4. Searching for the Archiso Disk

Code Snippet:

```
# Search for the ISO volume
if [ -z "${ARCHISO_UUID}" ]; then
    if [ -z "${ARCHISO_HINT}" ]; then
        regexp --set=1:ARCHISO_HINT '^\(([^)]+)\)' "${cmdpath}"
    fi
    search --no-floppy --set=root --file '%ARCHISO_SEARCH_FILENAME%' --hint
"${ARCHISO_HINT}"
    probe --set ARCHISO_UUID --fs-uuid "${root}"
fi
```

This section is dedicated to finding the medium from which GRUB was booted, which is presumed to contain the Arch Linux installation files (Archiso).

- `if [ -z "${ARCHISO_UUID}" ]`: This checks if the `ARCHISO_UUID` variable is empty, which means the Archiso disk has not yet been identified.
- `if [ -z "${ARCHISO_HINT}" ]`: This checks if there is no pre-existing hint.
- `regexp --set=1:ARCHISO_HINT '^\(([^)]+)\)' "${cmdpath}"`: This attempts to extract a hint (such as a path or filename) from the `cmdpath` variable, which represents the path from which GRUB was invoked.
- `search --no-floppy --set=root --file '%ARCHISO_SEARCH_FILENAME%' --hint "${ARCHISO_HINT}"`: This searches for a specific file (`%ARCHISO_SEARCH_FILENAME%` - which will be replaced with the actual filename at boot) on the disks, using the provided hint if available. When the file is found, the

  `root` variable is set to point to that device.

- `probe --set ARCHISO_UUID --fs-uuid "${root}"`: This probes the device identified by `${root}` to get the file system's **UUID (Universally Unique Identifier)** and stores it

in the `ARCHISO_UUID` variable. This UUID is later used to reliably identify the Archiso disk.

---

5. Default Menu and Timeout Settings

Code Snippet:

```
# Set default menu entry
default=archlinux
timeout=15
timeout_style=menu
```

- `default=archlinux`: This specifies that the first menu entry (the one with the `id` equal to `archlinux`) will be the default entry to be booted after the timeout expires.
- `timeout=15`: This sets the waiting time in seconds before the default option is automatically booted.
- `timeout_style=menu`: This specifies that the countdown will be displayed in a menu style.

---

6. GRUB Init Tune (for Accessibility)

Code Snippet:

```
# GRUB init tune for accessibility
play 600 988 1 1319 4
```

- `play frequency duration volume`: This command plays an audio tone. This line may be used to inform the user that GRUB has started, which is useful for visually impaired users. The numbers represent the frequencies, duration, and volume levels.

---

7. Defining Menu Entries

This is where the options that will appear to the user upon system boot are defined.

7.1. Arch Linux Installer Entry

Code Snippet:

```
menuentry "Helwan installer(x86_64, UEFI)" --class arch --class gnu-linux --
class gnu --class os --id 'archlinux' {
    set gfxpayload=keep
```

```
    linux /%INSTALL_DIR%/boot/x86_64/vmlinuz-linux
archisobasedir=%INSTALL_DIR% archisodevice=UUID=${ARCHISO_UUID}
cow_spacesize=5G nouveau.modeset=1 radeon.modeset=1 i915.modeset=1 copy-
toram=n nvme_load=yes
    initrd /%INSTALL_DIR%/boot/intel-ucode.img /%INSTALL_DIR%/boot/amd-
ucode.img /%INSTALL_DIR%/boot/x86_64/initramfs-linux.img
}
```

- `menuentry "..." --class ... --id 'archlinux'`: This command starts the definition of a menu entry with the name "Helwan installer(x86_64, UEFI)". The

  `--class` and `--id` are used to categorize the entry and identify it later.

- `set gfxpayload=keep`: This preserves any previous graphics settings.
- `linux /%INSTALL_DIR%/boot/x86_64/vmlinuz-linux ...`: This is the main command to boot the **Linux kernel**.
  - `/%INSTALL_DIR%/boot/x86_64/vmlinuz-linux`: The path to the kernel file.
  - **Kernel Parameters**:
    - `archisobasedir=%INSTALL_DIR%`: Defines the base directory for Archiso files.
    - `archisodevice=UUID=${ARCHISO_UUID}`: Specifies the Archiso device using the UUID found previously.
    - `cow_spacesize=5G`: Sets the size of the **Copy-on-Write (COW)** space to 5 GB.
    - `nouveau.modeset=1`, `radeon.modeset=1`, `i915.modeset=1`: Activate **modesetting** for Nvidia (nouveau), AMD (radeon), and Intel (i915) graphics cards.
    - `copytoram=n`: Prevents the file system from being copied to RAM.
    - `nvme_load=yes`: Ensures that NVMe support (for modern SSDs) is loaded early.
- `initrd /%INSTALL_DIR%/boot/intel-ucode.img /%INSTALL_DIR%/boot/amd-ucode.img /%INSTALL_DIR%/boot/x86_64/initramfs-linux.img`: This loads the **Initramfs (Initial RAM File System)**.
  - `intel-ucode.img` and `amd-ucode.img`: Microcode update files for Intel and AMD processors, to improve stability and performance.
  - `initramfs-linux.img`: The main initramfs file containing the tools necessary for system startup.

7.2. UEFI-Specific Entries

This section only appears if GRUB is running in UEFI mode.

- `if [ "${grub_platform}" == "efi" ]`: This checks if GRUB is running on a UEFI system.
- `if [ "${grub_cpu}" == "x86_64" ]`: If the system's processor is 64-bit:
  - `menuentry "Run Memtest86+ (RAM test)" ...`: An entry to run Memtest86+, a tool for testing RAM.

    - `linux /boot/memtest86+/memtest.efi`: This runs the Memtest86+ application in EFI format.
    - `menuentry "UEFI Shell" ...`: An entry to run the UEFI Shell, a command-line interface provided by the UEFI environment.
        - `insmod chain`: Loads the `chain` module, which allows "chaining" the load of another program.
        - `chainloader /shellx64.efi`: Loads and runs the 64-bit UEFI Shell application.
- `elif [ "${grub_cpu}" == "i386" ]`: If the processor is 32-bit (which is rare on modern UEFI systems):
    - `chainloader /shellia32.efi`: Loads and runs the 32-bit UEFI Shell application.
- `menuentry 'UEFI Firmware Settings' ...`: An entry that allows the user to access **UEFI firmware settings** directly from the GRUB menu.
    - `fwsetup`: The GRUB command for accessing UEFI settings.

7.3. Shutdown and Restart Entries

- `menuentry "System shutdown" ...`: An entry to shut down the system.
    - `halt`: The GRUB command to shut down the system.
- `menuentry "System restart" ...`: An entry to restart the system.
    - `reboot`: The GRUB command to restart the system.

Conclusion

This

`grub.cfg` file is designed to be bootable from a medium (like a USB drive or DVD) containing an Archiso environment. It loads the necessary modules, prepares the user interface (graphical and serial), searches for the Archiso environment, and then displays a list of boot options including the Arch Linux installer, useful tools like Memtest86+ and the UEFI Shell, as well as shutdown and restart options.

Explanation of the `loopback.cfg` File for GRUB: A Professional Technical Guide

This file,

`loopback.cfg`, is an integral part of the GRUB (Grand Unified Bootloader) boot system. It is responsible for providing the list of options you see when booting an operating system from a bootable medium such as a USB drive or DVD. This file specifically focuses on how to discover and load the operating system from an

ISO image.

Mechanism and Command Breakdown

The

`loopback.cfg` aims to locate the ISO image of the operating system (in this case, "Helwan install medium") and then configure GRUB to boot that system. Let's detail the main commands:

Searching for the ISO Medium

- `search --no-floppy --set=archiso_img_dev --file "${iso_path}"`: This command searches for the ISO image file specified in the `${iso_path}` variable. The

  `--no-floppy` option tells GRUB to ignore floppy drives, and `--set=archiso_img_dev` sets an internal variable named `archiso_img_dev` to point to the device that was found.

- `probe --set archiso_img_dev_uuid --fs-uuid "${archiso_img_dev}"`: After finding the device, this command probes it to get its unique file system **UUID** and stores it in the `archiso_img_dev_uuid` variable. This is necessary to ensure that GRUB can always reliably locate the ISO partition, even if the disk order changes.

Determining the Platform Identifier

- The commands that begin with

  `if [ "${grub_platform}" == ... ]` build an `archiso_platform` variable to accurately describe the current boot environment.

- If

  `grub_platform` is `'efi'` (UEFI), it checks the processor architecture (`grub_cpu`) and adds `'x64'` (for x86_64) or `'IA32'` (for i386) or simply the processor name if it's different.

- If

  `grub_platform` is `'pc'`, the platform is identified as `'BIOS'`.

- In other cases, a combination of

  `grub_cpu` and `grub_platform` is used to create a unique identifier. This ensures that the correct boot options are displayed based on whether the system is using UEFI or BIOS, and the processor architecture.

Default Menu Settings

- `default=archlinux`: This line specifies that the first option in the menu (which has the ID `archlinux`) will be the default, and it will be booted automatically after the timeout expires.
- `timeout=15`: This sets the wait time before booting the default option to 15 seconds.
- `timeout_style=menu`: This specifies that the timeout countdown will be displayed in a menu style.

Defining Menu Options

Each section that begins with

`menuentry "..." { ... }` represents a single option in the GRUB menu.

- `menuentry "Helwan install medium (%ARCH%, ${archiso_platform})"`: This is the main entry for installing the system.
  - `--class arch --class gnu-linux --class gnu --class os`: These classes are used to categorize and format the menu, and are often used by GRUB's graphical interfaces.
  - `--id 'archlinux'`: A unique identifier for this option, which is used in the `default=archlinux` line.
  - `set gfxpayload=keep`: This preserves the current graphics mode if one exists.
  - `linux /%INSTALL_DIR%/boot/%ARCH%/vmlinuz-linux ...`: This command loads the operating system's kernel (`vmlinuz-linux`).
    - `archisobasedir=%INSTALL_DIR%`: Specifies the base directory that contains the installation files inside the ISO image.
    - `img_dev=UUID=${archiso_img_dev_uuid}`: Passes the ISO device's UUID to the kernel.
    - `img_loop="${iso_path}"`: Passes the path of the ISO file to the kernel.
  - `initrd /%INSTALL_DIR%/boot/%ARCH%/initramfs-linux.img`: This loads the initial initramfs image, which is essential for starting the system.
- `menuentry "Helwan install medium with speakup screen reader ..."`: Similar to the first option, but it adds the `accessibility=on` parameter to the kernel line to enable the `speakup` screen reader, making it suitable for users with special needs.
- **Memtest86+ Options**: Options for Memtest86+ (a tool for testing RAM) are provided based on whether the system is running UEFI or BIOS.
- **UEFI Shell Option**: If the system is running UEFI, an option to access the UEFI Shell is provided, which allows direct execution of UEFI commands.
- **UEFI Firmware Settings Option**: This option allows the user to directly access the **UEFI firmware settings** (BIOS).
- **Shutdown and Restart Options**: `menuentry "System shutdown"` performs a system shutdown , while

  `menuentry "System restart"` performs a system restart.

Context and Practical Application

This file is a practical example of how to customize GRUB for specific boot environments, especially those that rely on ISO images. The

`loopback.cfg` file is used in Linux distributions based on Arch Linux (like Manjaro or Arch Linux itself when created from an ISO) to provide a smooth user experience when booting from removable media.

Professional Technical Summary

- **Flexibility**: The file supports booting via both **UEFI** and **BIOS**, as well as both 64-bit and 32-bit processors.
- **Automatic Detection**: It automatically detects the ISO's location and its UUID, making it resilient to changes in disk order.
- **Customization**: It allows for the definition of multiple options, including an accessibility mode and helpful system tools (Memtest, UEFI Shell).
- **Scalability**: The options can be easily modified to add more menu entries or change boot behavior.

2.6 syslinux/

```
 syslinux/
├──  archiso_head.cfg
├──  archiso_pxe-linux.cfg
├──  archiso_pxe.cfg
├──  archiso_sys-linux.cfg
├──  archiso_sys.cfg
├──  archiso_tail.cfg
├──  splash.png
│
└──  syslinux.cfg
```

The `syslinux` folders are a fundamental part of the system boot process, especially for systems that rely on SYSlinux or ISOLINUX. This type of boot loader is commonly used to boot operating systems from removable media like USB drives or over a network using PXE.

The folder you provided contains the configuration files and graphical interfaces necessary for booting an Arch Linux system from bootable media.

Here is a detailed explanation of each file:

- **`syslinux.cfg`**: This is the main SYSlinux configuration file. It acts as the entry point, defining the default boot menu, timeout, graphical background, and other configuration files to be included. This file typically calls additional configuration files like `archiso_sys.cfg` and `archiso_tail.cfg` to organize the settings.
- **`archiso_sys.cfg`**: This file is dedicated to boot settings for Arch Linux systems using BIOS. It contains menu entries that allow the user to run the Arch Linux installer or other tools like Memtest86+ in a BIOS environment.
- **`archiso_head.cfg`**: This file defines the beginning of the configuration files. It includes general commands and a preamble that are included in other configuration files, which prevents code repetition.
- **`archiso_tail.cfg`**: This file represents the end of the configuration files. It contains final menu entries such as shutdown and reboot options, which always appear at the bottom of the boot menu.
- **`archiso_pxe.cfg`**: This file is dedicated to booting the system over a network using **PXE (Preboot Execution Environment)**. It provides boot options for clients connecting to a PXE server to boot the Arch Linux installer without the need for physical media.
- **`archiso_pxe-linux.cfg`**: This file is used to define the menu entries that are displayed to PXE clients. It complements `archiso_pxe.cfg` by providing the specific instructions to load the kernel and initrd files over the network.
- **`archiso_sys-linux.cfg`**: This file is similar to `archiso_pxe-linux.cfg`, but it is used for boot settings from local media (like a USB) in a BIOS environment.

- **`splash.png`**: This is a PNG image used as the graphical background for the boot menu. It provides an attractive visual interface instead of the default text-based screen.

The idea behind separating the configuration files is to create a modular structure that is easy to manage. Instead of having one large file, the settings are divided into smaller parts, each with a specific purpose (such as common settings, BIOS settings, or PXE settings). This approach facilitates customization and maintenance.

When you are ready, send the content of each file, and I will integrate it into this explanation in a professional manner.

`syslinux.cfg`: The Main SYSlinux Configuration File

This file acts as the primary entry point for the SYSlinux bootloader. It defines the overall structure of the boot menu and directs the system to the appropriate sub-configuration files based on the boot environment. This modular structure is essential for maintaining organized and clear settings.

- `DEFAULT select`: This command specifies that the default option to be selected in the boot menu is **`select`**.
- `LABEL select`: This line defines the boot menu entry named `select`.
- `COM32 whichsys.c32`: This command loads the `whichsys.c32` module, a utility specifically developed for Archiso. Its main function is to examine the current boot environment to determine whether the boot is from local media or over the network (PXE).
- `APPEND -pxe- pxe -sys- sys -iso- sys`: This command passes parameters to the `whichsys.c32` module. These parameters are used to guide the module to specific "platforms."
    - `-pxe- pxe`: If a PXE boot is detected, it will redirect to the `pxe` label.
    - `-sys- sys`: If a boot from local media (like a USB drive) is detected, it will redirect to the `sys` label.
    - `-iso- sys`: This parameter ensures that booting from an ISO image also goes to the `sys` label.
- `LABEL pxe`: This label defines the starting point for PXE boot settings.
- `CONFIG archiso_pxe.cfg`: This command directs SYSlinux to load the `archiso_pxe.cfg` configuration file, which contains all the options and settings for network booting.
- `LABEL sys`: This label defines the starting point for local media boot settings.
- `CONFIG archiso_sys.cfg`: This command directs SYSlinux to load the `archiso_sys.cfg` configuration file, which contains the boot options specific to booting from a USB drive or DVD.

In summary, the `syslinux.cfg` file functions as an intelligent control panel, using `whichsys.c32` to automatically determine the operating environment and then passing control to the correct configuration file, providing a smooth and efficient boot experience for the user.

`archiso_sys.cfg`: The BIOS Boot Configuration

This file serves as the main hub for boot settings for Arch Linux systems that are booted from local media in a **BIOS (Basic Input/Output System)** environment. Its primary role is to call other configuration files in an organized manner to create a complete and integrated boot menu.

- `INCLUDE archiso_head.cfg`: This command instructs SYSlinux to include the content of the `archiso_head.cfg` file. As previously mentioned, this file contains general and introductory settings that are used across all configuration files. This practice ensures consistency and prevents code duplication.
- `DEFAULT arch64`: This command specifies that the default option to be selected in the boot menu is **`arch64`**.
- `TIMEOUT 150`: This line sets the wait time in centiseconds (1/100 of a second) before the default option is automatically booted. The value `150` is equivalent to 1.5 seconds, giving the user enough time to choose a different option.
- `INCLUDE archiso_sys-linux.cfg`: This is a fundamental command that instructs SYSlinux to include the content of the `archiso_sys-linux.cfg` file. This file contains the actual menu entries, such as the option to install Arch Linux.
- `INCLUDE archiso_tail.cfg`: This command represents the final step in building the boot menu, as it includes the content of the `archiso_tail.cfg` file, which contains the final options like shutdown and reboot.

In short, `archiso_sys.cfg` acts as a coordinator that combines different parts of the configuration to create a complete boot menu tailored for BIOS environments, ensuring that all necessary options are available to the user in a logical sequence.

`archiso_head.cfg`: The Common Configuration Header

This file serves as a **common header** for various SYSlinux configuration files. Its primary purpose is to define global settings and visual elements that are shared across different boot environments (e.g., local media, PXE). By centralizing these configurations, it ensures consistency and avoids code duplication, adhering to best practices in modular design.

The file's content can be broken down into three main sections: **console and UI setup**, **menu dimensions and layout**, and **color scheme**.

Console and UI Setup

- `SERIAL 0 115200`: This command configures a **serial console**. It activates the serial port 0 (typically COM1) and sets the communication speed to **115,200 baud**. This is essential for remote system management and debugging, as it allows administrators to interact with the bootloader from a remote terminal.
- `UI vesamenu.c32`: This loads the **VesaMenu** module, a graphical user interface (GUI) for SYSlinux. It replaces the default text-based menu with a more visually appealing one, enabling the use of backgrounds and advanced menu customization.
- `MENU TITLE Helwan`: This sets the **title** that appears at the top of the boot menu, in this case, "Helwan."
- `MENU BACKGROUND splash.png`: This specifies the **background image** for the boot menu, pointing to the `splash.png` file. This provides a professional and branded look for the bootloader.

Menu Dimensions and Layout

This section defines the precise layout of the boot menu on the screen. The values are in characters, allowing for fine-tuned control over the menu's appearance.

- `MENU WIDTH 78`: Sets the width of the main menu window to 78 characters.
- `MENU MARGIN 4`: Sets the left margin of the menu to 4 characters.
- `MENU ROWS 7`: Defines the number of visible menu rows.
- `MENU VSHIFT 10`: Vertically shifts the menu down by 10 rows.
- `MENU TABMSGROW 14`, `MENU CMDLINEROW 14`, `MENU HELPMSGROW 16`, `MENU HELPMSGENDROW 29`: These commands set the vertical positions for the tab message, command line, and help message areas, ensuring they don't overlap with the main menu items.

Color Scheme

This is a critical section for visual customization. It defines the color of every element in the boot menu using a hexadecimal color format.

- `MENU COLOR border 30;44 #40ffffff #a0000000 std`: Configures the color of the menu border. The values represent foreground color, background color, and specific hexadecimal codes for transparency and color blending.

- The subsequent `MENU COLOR` lines (e.g., `title`, `sel`, `unsel`) follow the same format, allowing for detailed control over the colors of the title, selected items, unselected items, help text, and other menu components.

Menu Behavior

- `MENU CLEAR`: This command clears the screen before the menu is displayed, ensuring a clean presentation.
- `MENU IMMEDIATE`: This command causes the menu to be displayed immediately without waiting for a user key press.

In summary, `archiso_head.cfg` is not a bootable file on its own. It's a foundational component that's included by other configuration files to establish a consistent, well-designed, and highly functional boot environment. Its role is to define the "look and feel" and basic behavior of the SYSlinux menu, separating presentation from functionality.

`archiso_tail.cfg`: Common Utility Menu Entries

This file contains the final set of utility and system management options that are typically appended to the end of the boot menu. These entries provide essential functions for users interacting with the bootable medium, offering ways to boot installed systems, perform hardware diagnostics, and manage the system's power state.

- `LABEL existing`
  - **Purpose**: This entry allows the user to boot an **existing operating system** that is already installed on the computer's hard drive. This is crucial for live environments that also serve as boot managers for installed systems.
  - `TEXT HELP ... ENDTEXT`: This block provides helpful information to the user. It clarifies that this option boots an installed OS and instructs the user to press `TAB` to edit boot parameters (disk and partition numbers).
  - `MENU LABEL Boot existing OS`: Sets the visible text for this menu item.
  - `COM32 chain.c32`: Loads the `chain.c32` module, which is responsible for **chainloading** another bootloader (typically the one from the installed OS's boot sector).
  - `APPEND hd0 0`: This parameter tells `chain.c32` to boot from the first hard disk (`hd0`) and the first partition (`0`) on that disk. This is a common way to specify the primary boot partition of an installed system.

- `LABEL memtest`
  - **Purpose**: This entry initiates **Memtest86+**, a widely used tool for **testing the system's RAM (Random Access Memory)**. Performing a RAM test is a fundamental diagnostic step to rule out memory errors as the cause of system instability or performance issues.
  - `MENU LABEL Run Memtest86+ (RAM test)`: Sets the visible text for this menu item, clearly indicating its function.
  - `LINUX /boot/memtest86+/memtest`: This command loads the Memtest86+ executable file, which is located within the `/boot/memtest86+` directory on the boot medium.

- `LABEL hdt`
  - **Purpose**: This entry launches **HDT (Hardware Detection Tool)**, a utility that provides detailed information about the computer's hardware components. This is invaluable for troubleshooting hardware compatibility issues or simply understanding the system's configuration.
  - `MENU LABEL Hardware Information (HDT)`: Sets the visible text for this menu item.
  - `COM32 hdt.c32`: Loads the HDT module.

    - `APPEND modules_alias=hdt/modalias.gz pciids=hdt/pciids.gz`: These parameters pass configuration files to HDT. `modules_alias.gz` helps HDT map hardware devices to the correct kernel modules, while `pciids.gz` provides a mapping of PCI device IDs for better identification.

- `LABEL reboot`
    - **Purpose**: This option allows the user to **reboot** the computer directly from the boot menu.
    - `TEXT HELP ... ENDTEXT`: The help text specifies that the computer's firmware must support **APM (Advanced Power Management)** for this option to function correctly.
    - `MENU LABEL Reboot`: Sets the visible text for this menu item.
    - `COM32 reboot.c32`: Loads the `reboot.c32` module, which handles the system reboot process.

- `LABEL poweroff`
    - **Purpose**: This option allows the user to **power off** the computer directly from the boot menu.
    - `TEXT HELP ... ENDTEXT`: Similar to reboot, this help text notes that the computer's firmware must support **APM (Advanced Power Management)** for the power-off functionality.
    - `MENU LABEL Power Off`: Sets the visible text for this menu item.
    - `COM32 poweroff.c32`: Loads the `poweroff.c32` module, which gracefully shuts down the system and powers off the hardware.

In essence, `archiso_tail.cfg` consolidates essential system utilities into a user-friendly menu, complementing the core operating system installation or live environment options. These are often the last items presented, providing fallback and diagnostic capabilities.

`archiso_pxe.cfg`: Network Boot (PXE) Configuration

This file is an integral part of the boot setup process using the **PXE (Preboot Execution Environment)** protocol. Its primary goal is to provide the necessary boot options for a client device requesting to boot over the network, enabling it to load an Arch Linux operating system without requiring local storage media (like USB or DVD).

- `INCLUDE archiso_head.cfg`: This command includes the `archiso_head.cfg` file. As previously explained, this file provides common, shared settings and environmental configurations, such as the GUI (VesaMenu) settings, title, and background image (`splash.png`). This inclusion ensures that all devices booted via PXE share the same basic look and initial setup.
- `INCLUDE archiso_pxe-linux.cfg`: This is the pivotal command within this file. It includes the `archiso_pxe-linux.cfg` file, which contains the **actual menu entries** for PXE booting. Here, the paths to the Linux kernel and the `initramfs` image to be loaded over the network are defined, along with any necessary kernel parameters for network booting.
- `INCLUDE archiso_tail.cfg`: This command completes the final build of the boot menu by including the `archiso_tail.cfg` file. This file typically contains the final boot options such as "Reboot" and "Power Off," which appear at the end of the menu regardless of the boot method (PXE or local).

In summary, the `archiso_pxe.cfg` file acts as a coordinator for the network boot process. It first applies common graphical and environmental settings from `archiso_head.cfg`, then loads the specific options for PXE booting from `archiso_pxe-linux.cfg`, and finally appends standard shutdown options from `archiso_tail.cfg`. This modular structure ensures a consistent and efficient network boot experience.

Network Boot Menu Options in `archiso_pxe-linux.cfg`

This file contains the specific menu entries that appear when booting a system over the network. It defines how the Linux kernel and initial system files (initramfs) are loaded from a network server using various protocols.

`LABEL arch64_nbd`

- **`TEXT HELP`** **and** **`ENDTEXT`**: Provides help text explaining that this option allows booting the Helwan installation medium via NBD (Network Block Device), enabling system installation or maintenance.
- **`MENU LABEL Helwan install medium (x86_64, NBD)`**: Specifies the text that will be displayed in the boot menu.
- **`LINUX ::/%INSTALL_DIR%/boot/x86_64/vmlinuz-linux`**: Defines the path to the Linux kernel.
- **`INITRD ::/%INSTALL_DIR%/boot/intel-ucode.img,::/%INSTALL_DIR%/boot/amd-ucode.img,::/%INSTALL_DIR%/boot/x86_64/initramfs-linux.img`**: Specifies the paths to the initrd files. The `::` prefix indicates that these files should be loaded over the network.
- **`APPEND archisobasedir=%INSTALL_DIR% archisodevice=UUID=%ARCHISO_UUID% archiso_nbd_srv=${pxeserver} cms_verify=y`**: Passes the necessary kernel parameters to set up the installation environment, including the NBD server.

`LABEL arch64_nfs`

- **`TEXT HELP`** **and** **`ENDTEXT`**: Provides help text explaining that this option allows booting the Helwan installation medium via NFS (Network File System), enabling system installation or maintenance.
- **`MENU LABEL Helwan install medium (x86_64, NFS)`**: Specifies the text that will be displayed in the boot menu.
- **`LINUX ::/%INSTALL_DIR%/boot/x86_64/vmlinuz-linux`**: Defines the path to the Linux kernel.
- **`INITRD ::/%INSTALL_DIR%/boot/intel-ucode.img,::/%INSTALL_DIR%/boot/amd-ucode.img,::/%INSTALL_DIR%/boot/x86_64/initramfs-linux.img`**: Specifies the paths to the initrd files.
- **`APPEND archisobasedir=%INSTALL_DIR% archiso_nfs_srv=${pxeserver}:/run/archiso/bootmnt cms_verify=y`**: Passes kernel parameters to specify the NFS server and the installation path.

`LABEL arch64_http`

- **`TEXT HELP`** **and** **`ENDTEXT`**: Provides help text explaining that this option allows booting the Helwan installation medium via HTTP, enabling system installation or maintenance.
- **`MENU LABEL Helwan install medium (x86_64, HTTP)`**: Specifies the text that will be displayed in the boot menu.

- **`LINUX ::/%INSTALL_DIR%/boot/x86_64/vmlinuz-linux`**: Defines the path to the Linux kernel.
- **`INITRD ::/%INSTALL_DIR%/boot/intel-ucode.img,::/%INSTALL_DIR%/boot/amd-ucode.img,::/%INSTALL_DIR%/boot/x86_64/initramfs-linux.img`**: Specifies the paths to the initrd files.
- **`APPEND archisobasedir=%INSTALL_DIR% archiso_http_srv=http://${pxeserver}/ cms_verify=y`**: Passes kernel parameters to specify the HTTP server.

Understanding `archiso_sys-linux.cfg` for BIOS Boot

This configuration file, `archiso_sys-linux.cfg`, defines the boot menu options for systems booting via **BIOS** in a network boot environment, specifically for the Helwan installation medium. It outlines how the Linux kernel and initial RAM disk (initramfs) are loaded to start the installation or maintenance process.

Here's a breakdown of each component:

- **`LABEL arch64`**: This is a unique identifier for this specific boot entry. It's how the bootloader refers to this particular configuration.
- **`TEXT HELP ... ENDTEXT`**: This block provides descriptive help text that appears when this menu option is selected. It informs the user that this option is for booting the Helwan install medium on a BIOS system and can be used for installing Helwan or performing system maintenance.
- **`MENU LABEL Helwan install medium (x86_64, BIOS)`**: This is the user-friendly text that will be displayed in the boot menu, allowing users to easily identify and select this option.
- **`LINUX /%INSTALL_DIR%/boot/x86_64/vmlinuz-linux`**: This directive specifies the location of the **Linux kernel** (`vmlinuz-linux`). The `/%INSTALL_DIR%/` part indicates that this kernel is part of the installation media, and it's located within the `boot/x86_64/` directory.
- **`INITRD /%INSTALL_DIR%/boot/intel-ucode.img,/%INSTALL_DIR%/boot/amd-ucode.img,/%INSTALL_DIR%/boot/x86_64/initramfs-linux.img`**: This directive specifies the initial RAM disk images that are loaded along with the kernel.
    - `intel-ucode.img` and `amd-ucode.img` are microcode updates for Intel and AMD processors, respectively. These help ensure proper hardware initialization and stability.
    - `initramfs-linux.img` is the main initial RAM file system, which contains essential drivers and tools needed to mount the root file system and continue the boot process.
- **`APPEND archisobasedir=%INSTALL_DIR% archisodevice=UUID=%ARCHISO_UUID% cow_spacesize=5G nouveau.modeset=1 radeon.modeset=1 i915.modeset=1 copytoram=n nvme_load=yes`**: This line passes various **kernel parameters** (or boot arguments) that configure the environment for the installation.
    - `archisobasedir=%INSTALL_DIR%`: Tells the system where the main Archiso installation files are located.
    - `archisodevice=UUID=%ARCHISO_UUID%`: Specifies the device containing the installation media using its **Universally Unique Identifier (UUID)**. This ensures the system can reliably find the installation source.
    - `cow_spacesize=5G`: Sets the size of the Copy-on-Write (CoW) space to 5 Gigabytes. This is used for temporary changes during the live environment.
    - `nouveau.modeset=1`, `radeon.modeset=1`, `i915.modeset=1`: These parameters enable kernel mode setting for Nvidia (`nouveau`), AMD (`radeon`), and Intel (`i915`) graphics drivers, respectively. This helps in getting proper graphics output early in the boot process.

- `copytoram=n`: Indicates that the installation media should **not** be copied entirely into RAM. This saves memory but means the system will access the installation files from their original location (e.g., network or USB).
- `nvme_load=yes`: Ensures that NVMe storage drivers are loaded, which is crucial for systems using NVMe SSDs.

In essence, this `archiso_sys-linux.cfg` entry provides a specific configuration for booting the Helwan installation environment on older BIOS systems, ensuring all necessary kernel modules and parameters are correctly set for a successful boot and installation process.

2.6.1 The Secrets of the /etc/ Directory

Introduction

When we talk about building a Linux distribution using archiso, developers often focus on choosing packages, configuring the desktop environment, or adding some helper scripts. However, the truth is that all these details, despite their importance, only scratch the surface. The essence, the true personality of the distribution, is planted in one place: the /etc/ directory.

This directory isn't just a "configuration bin" as some might imagine; it's the system's brain. It contains everything that defines the distribution's identity:

- How the system handles users and their permissions.
- How networks and services are managed.
- The distribution's language, time zone, and network name.
- How the system boots, and which kernel and drivers are loaded.
- Which repositories the package manager, **pacman**, trusts.
- How a user first logs into the desktop environment, and what default settings they receive.

All these details and more are stored in /etc/. This is where the danger of this directory lies: a small error inside it can prevent the system from booting, and a precise touch can change the user experience completely.

From UNIX to Helwan Linux: The Story of a Small Directory

Historically, the name /etc/ began as an abbreviation for "et cetera" (etc.), a place to collect files that didn't have a better home. But with the evolution of UNIX and then GNU/Linux, it gradually transformed into the system's center of gravity, until today it serves as the distribution's **"constitution."**

By carefully examining /etc/, one can understand the philosophy of any operating system: Is it geared towards developers? Beginners? Servers? Or a specific community, like the identity of Helwan Linux?

The Map: /etc/ in Helwan Linux

To simplify understanding, we can view /etc/ as a tree with branching roots. In the Helwan Linux distribution, this directory takes on a rich structure, as shown below: etc/ ├── User Management (passwd, shadow, group, gshadow) ├── System Identity (hostname, locale.conf, localtime) ├── Boot and Startup (grub, mkinitcpio*, modprobe.d) ├── pacman Package Manager (pacman.conf, pacman.d/hooks) ├── Graphical System (lightdm, skel, X11) ├── Networking

(resolv.conf, systemd/network, ssh) ├── Security and Permissions (sudoers.d, polkit, motd) ├── systemd and Services (network, display, cloud-init, zram …) └── Graphical User Settings (xdg/reflector, cinnamon configs)

These are not just file names; each element is a separate story. For example:

- **passwd** and **shadow** represent the backbone of account management.
- **pacman.conf** precisely defines where software comes from and how its integrity is verified.
- **lightdm.conf** and **slick-greeter.conf** configure the first screen the user sees.
- **reflector.service** ensures that mirrors are updated automatically for fast and stable installations and updates.
- The **skel/Templates/hel-files** directory reflects the unique fingerprint of Helwan Linux, offering ready-made files that facilitate the work of programmers and content creators.

Live vs. Installed: Two Personalities in One Entity

One of the cleverest details that appears within /etc/ is the contrast between the Live environment and the Installed environment:

- In the **Live** environment: Temporary settings, automatic login, open permissions, and services geared for a trial run.
- In the **Installed** environment: A new user with a password, tightened permissions, experimental services stopped, and the system transitioned to a secure, complete environment.

This contrast is no accident; it is carefully managed through files and services within /etc/.

The Goal of This Chapter

In this chapter, we won't just review the files; we will delve into their depths. We will explain what each file does, how it interacts with the rest of the system, and why it's important in the context of building a distribution. We will divide the journey into eight main axes:

1. User and security management.
2. System identity.
3. Boot and startup.
4. The **pacman** package manager.
5. Graphical system and login interface.
6. Networking and communications.
7. Security and permissions.
8. **systemd** and services.
9. Graphical user settings.

We will focus on over sixty files and directories, some of which are fundamental to the system's operation, while others add a special touch to the identity of Helwan Linux.

Conclusion of the Introduction

Our journey into /etc/ is not just a lesson in text files; it's a journey to discover the distribution's personality from the inside. Anyone who reads this directory in depth will realize that every file, no matter how simple, is a thread in a complete fabric.

Now, let's start from the beginning: user and security management, where the first rules of the game are written.

2.6.2 User and Security Management

/etc/passwd File

This file is the **backbone** for defining users in any Unix/Linux system. Each line represents a user, and the fields within are separated by ::.

root:x:0:0:root:/root:/usr/bin/zsh

liveuser:x:1000:1000::/home/liveuser:/bin/bash

- **root**: The user with the highest privileges (superuser).
- **UID = 0**: Absolute permissions.
- **shell = zsh** instead of bash: A design decision in Helwan Linux for a more modern experience.
- **liveuser**: The user for the Live session.
- **UID = 1000**: The first regular user.
- **shell = bash**: Simple and easy for beginners.

 Philosophy: In Unix, "everything is a file." Even user definitions are just text that can be read and modified.

/etc/shadow File

This file stores **passwords** (in an encrypted form).

root::14871::::::

liveuser::14871::::::

The empty fields mean the accounts have no passwords. **Result**: Direct login in the Live environment without needing a password.

/etc/group File

Groups define **shared permissions**.

wheel:x:998:liveuser

network:x:90:liveuser

audio:x:995:liveuser

video:x:986:liveuser

storage:x:988:liveuser

- **wheel**: For sudo access.
- **network**: For network management.
- **audio/video/storage**: For media playback and storage.

 Result: liveuser is immediately ready for a full experience without restrictions.

/etc/gshadow File

This file carries the same idea as /etc/group but for **authentication**.

wheel:!!::liveuser

- **!!**: Indicates no password is set for the group.

/etc/sudoers.d/g_wheel File

%wheel

ALL=(ALL) NOPASSWD: ALL

Any member of the wheel group (like liveuser) can execute sudo commands without a password. This is useful for the Live environment but is typically changed after installation to secure the system.

2.6.3. General System Settings

/etc/hostname

archiso

This file sets the **default hostname** for the system. In this case, archiso is a simple, clear name indicating that the system is built upon ArchISO. It's easily changeable during the installation process.

/etc/locale.conf

LANG=C.UTF-8

This configuration specifies the **system's locale**. C.UTF-8 is a **neutral, internationally compatible choice** that effectively prevents character encoding issues. It ensures consistent output from programs and logs across the system.

/etc/localtime

/usr/share/zoneinfo/UTC

This entry points to the system's **time zone**. By default, it's set to Coordinated Universal Time (UTC). This **simplifies global installation** and allows for **time zone customization** at a later stage.

2.6.4. Boot and Initialization Management

/etc/default/grub

This file provides **complete control over the GRUB bootloader**.

- GRUB_DISTRIBUTOR="Helwan": Defines the **visual identity** of the distribution within the GRUB menu.
- GRUB_CMDLINE_LINUX_DEFAULT="loglevel=7 audit=0":
  - loglevel=7: **Displays detailed messages** during boot, which is crucial for **diagnosing errors**.
  - audit=0: **Disables the auditing system** to **speed up the boot process**.
- GRUB_THEME="/usr/share/grub/themes/helwan/theme.txt": Applies a **custom theme** for a **visually appealing and professional look**.
- GRUB_DISABLE_OS_PROBER=false: **Enables OS-Prober**, which is useful for **detecting other operating systems** and facilitating dual-boot setups.

/etc/mkinitcpio.conf

This file configures the **initial RAM disk (initramfs)**, which contains essential modules loaded before the main root filesystem.

- HOOKS=(base udev modconf kms memdisk archiso archiso_loop_mnt archiso_pxe_common archiso_pxe_nbd archiso_pxe_http archiso_pxe_nfs block filesystems keyboard): These **hooks determine what is loaded into the initramfs**. The archiso hooks are specifically for running the system from an ISO or PXE boot. The keyboard hook ensures **keyboard support in emergency situations**.
- COMPRESSION="xz": Uses **XZ compression**, resulting in a **smaller ISO size**.

/etc/mkinitcpio.d/linux.preset

This file **specifies the primary image used for booting**.

- archiso_image="/boot/initramfs-linux.img": Points to the **main initramfs image** for the system.

/etc/mkinitcpio.conf.d/archiso.conf

This is a specialized configuration for Archiso.

- HOOKS=(base udev microcode modconf kms memdisk archiso ...): Includes the microcode hook for **processor microcode updates**, enhancing stability and security.
- COMPRESSION="xz" and COMPRESSION_OPTIONS=(-9e): Utilizes **very high compression** to significantly **reduce the ISO size**.

/etc/modprobe.d/broadcom-wl.conf

This configuration file **prevents conflicts with Broadcom wireless drivers**.

- # The broadcom-wl package requires some modules to be disabled...: Comments indicating that certain modules are **blacklisted to avoid conflicts** with the Broadcom proprietary wireless driver. This ensures that **WiFi functionality is available immediately upon first boot**.

2.6.5. The pacman Package Manager

The **pacman** package manager is the backbone of software management in Arch Linux and its derivatives, such as Helwan Linux. Its role extends beyond merely installing software; it also encompasses system updates, the removal of unwanted packages, and the verification of package integrity through digital signatures.

4.1 pacman.conf — The Control Center

The file **/etc/pacman.conf** is responsible for defining how **pacman** operates. Through it, **Repositories**, security policies, and package downloading mechanisms are configured.

Key Sections:

- **General Options [options]**
  - HoldPkg = pacman glibc Ensures that essential packages like **pacman** and **glibc** are not inadvertently removed or updated.
  - Architecture = auto Specifies the target architecture (e.g., x86_64, arm). The auto value allows **pacman** to determine this automatically.
  - ParallelDownloads = 5 Allows up to 5 packages to be downloaded concurrently, accelerating the process.
  - SigLevel = Required DatabaseOptional Defines the digital signature verification policy: packages must be signed, while repository databases have optional signature verification.
  - LocalFileSigLevel = Optional When installing packages from local files, signature verification is optional.
- **Repositories [repositories]**
  - **[core]** and **[extra]**: The official, essential repositories from Arch Linux.
  - **[helwan]**: A dedicated repository for the Helwan Linux distribution, which includes custom packages tailored for the distribution.
  - [helwan]
  - SigLevel = Optional TrustedOnly
  - Server = https://helwan-linux.github.io/$repo/$arch

Here, we observe that signature verification is optional, but packages must originate from a trusted source.

- **Note:** Other repositories like multilib or custom can be added, but they are disabled by default.

4.2 pacman Hooks — Automation

In addition to configuring **pacman.conf**, the behavior of **pacman** after operations can be controlled using hooks. The directory **/etc/pacman.d/hooks/** contains small scripts that execute automatically after specific packages are installed, upgraded, or removed.

Examples of Hooks in Helwan Linux:

- **uncomment-mirrors.hook**
  - **Trigger**: Executes after the **pacman-mirrorlist** package is installed or updated.
  - **Action**: Uses the sed command to uncomment all servers in **/etc/pacman.d/mirrorlist**, effectively enabling all mirrors automatically.
  - **Objective**: To ensure the system has an active mirror list immediately upon booting from the ISO.
- **zzzz99-remove-custom-hooks-from-airootfs.hook**
  - **Trigger**: Executes for any operation (Install, Upgrade, Remove) on any package.
  - **Action**: Executes a script that deletes any hook containing the phrase "remove from airootfs."
  - **Objective**: These hooks are necessary only during the ISO build process (live environment) but should not remain active on the installed system.

 With this mechanism, we can customize **pacman**'s behavior in the ISO environment without affecting the system after installation.

4.3 Conclusion

- **pacman.conf** defines **pacman**'s policies (security, repositories, download options).
- **Hooks** provide automation for post-installation or upgrade processes.
- Integrating the dedicated Helwan Linux repository makes the distribution self-contained, distinct from Arch, while retaining the robustness of the base system.

2.6.6 Mirror Management

When using pacman, download speed and service quality primarily depend on selecting the appropriate server (mirror). Helwan Linux provides a ready-made mechanism for choosing the best mirrors using the reflector tool.

2.6.6.1 reflector Configuration File

Location: /etc/xdg/reflector/reflector.conf

This file dictates how the reflector tool operates when updating the mirror list. It specifies criteria such as:

- **--country**: Defines the country or group of countries closest to the user.
- **--protocol**: The download protocol (http/https).
- **--sort**: How servers are ordered (e.g., by speed).
- **--latest N**: Uses the N most recently tested servers.

Realistic Example:

Bash

--country Egypt,France,Germany \

--protocol https \

--latest 20 \

--sort rate \

--save /etc/pacman.d/mirrorlist

This ensures that the mirrorlist is continuously updated with the best servers, significantly accelerating package downloads.

2.6.6.2 reflector Systemd Service

Location: /etc/systemd/system/reflector.service

This service is integrated into Helwan Linux to run on every system boot or periodically via a timer. Its function is to:

- Execute the reflector tool using the aforementioned configuration file.
- Automatically update /etc/pacman.d/mirrorlist.
- Guarantee that pacman always uses the fastest and most up-to-date mirrors without manual intervention.

Powerful Advantage for End-Users: Users don't need to manually enter and modify mirrors, which was often a point of frustration for beginners on Arch Linux.

2.6.6.3 Pacman Services and Hooks Integration

Helwan Linux not only relies on pacman.conf and mirrors but also utilizes a combination of systemd services and pacman hooks.

- **pacman hooks** (as seen previously) clean up and modify configurations during package installation, upgrades, or removals.
- The **reflector service** automatically updates mirrors.

This setup ensures the system remains fast, secure, and configured from day one.

2.6.6.4 Conclusion

- **pacman.conf**: The control center for the package manager.
- **pacman hooks**: Intelligent automation during installation/upgrades.
- **reflector**: Automatic selection of the best servers.
- **systemd services**: Guarantees continuous mirror updates without user input.

Helwan Linux combines all these elements to deliver a ready-to-use version of Arch Linux, eliminating the burdensome manual configuration that often deters most users.

2.6.7 pacman-init.service: Initializing pacman on First Boot

Location: /etc/systemd/system/pacman-init.service

2.6.7.1 Purpose of the Service

The pacman package manager relies on a GPG keyring mechanism to ensure that added or installed packages are digitally signed and originate from trusted sources (Arch Linux or Helwan repositories). However, in an ISO image or a fresh installation, the **keyring** is not initialized. This is where this service comes into play:

- It initializes the keyring (creates the key database for the first time).
- It populates the keyring with the keys of trusted entities (official Arch packagers + Helwan repo).

2.6.7.2 Explanation of File Sections

[Unit]

- **Description**: Clarifies the purpose (keyring initialization).
- **Requires=etc-pacman.d-gnupg.mount**: The service depends on the /etc/pacman.d/gnupg/ directory, which stores the keys. It must be available before execution.
- **After=... time-sync.target**: Depends on time synchronization because digital signature verification requires a correct timestamp (for certificate validity).
- **Before=archlinux-keyring-wkd-sync.service**: Must run before the auxiliary key synchronization service.

[Service]

- **Type=oneshot**: The service runs only once during boot.
- **RemainAfterExit=yes**: It considers itself active even after it finishes, so any subsequent services perceive the key as initialized.
- **ExecStart=/usr/bin/pacman-key --init**: Creates a new keyring.
- **ExecStart=/usr/bin/pacman-key --populate**: Adds Arch and Helwan keys to the store.

[Install]

- **WantedBy=multi-user.target**: This means it will be enabled by default in the standard operating mode (multi-user).

2.6.7.3 Why is This Important?

Without keyring initialization, pacman will refuse to install or update any signed packages. This protects the system from any tampering or the download of untrusted packages. The presence of

this service makes the **Helwan Linux ISO** ready from the first boot, without the user having to manually execute commands like:

Bash

pacman-key --init

pacman-key --populate

2.6.7.4 Added Value in Helwan Linux

In a standard Arch distribution, the user must manage the key initialization process themselves. In Helwan Linux, this service is integrated **out-of-the-box** to:

- Save setup time.
- Ensure complete security from day one.
- Provide a seamless experience, even for non-experts.

2.6.8 The /etc/xdg/reflector/reflector.conf File

This file is the core of the reflector service, the tool responsible for periodically updating pacman's mirrorlist file to select the fastest available servers. In a distribution built with archiso like Helwan Linux, this file ensures users get a fast and stable download experience from the very first moment.

File Contents:

Reflector configuration file for the systemd service.

--save /etc/pacman.d/mirrorlist

--ipv4

--ipv6

--protocol https

--latest 20

--sort rate

Detailed Explanation:

- --save /etc/pacman.d/mirrorlist Any updates to the Arch Linux or Helwan Linux servers are directly saved to this file. In other words, pacman will always read from the automatically generated mirrorlist. This provides users with speed in installing and updating packages from the nearest and fastest servers.
- --ipv4 and --ipv6 These options allow reflector to fetch servers that support both protocols. This is crucial for the distribution to function in different environments, whether an older local network (IPv4) or modern global networks and servers (IPv6).
- --protocol https This forces reflector to use secure servers (HTTPS). The goal is to protect against Man-in-the-Middle (MITM) attacks and ensure that packages are downloaded securely from trusted mirrors.
- --latest 20 This means reflector will fetch the 20 most recently updated servers from the official mirror list. Servers that are not updated are automatically removed.
- --sort rate This selects the fastest servers based on their download rate. This ensures that users of Helwan Linux find their updates much faster than relying on a random mirrorlist.

Importance of the File within the Distribution:

- **Creates a Professional User Experience:** As soon as a user launches the Live environment or after installation, they experience very high speed in package downloads.

- **Reduces Issues:** It minimizes "time out" errors or slow server problems that could create a poor first impression of the distribution.
- **Provides Smart Defaults:** It gives Helwan Linux an intelligent configuration from the start, rather than forcing the user to manually search for mirrors

2.6.9 LightDM: The Graphical Display Manager

LightDM is the component responsible for the login screen in the Helwan Linux distribution. Its main configuration file is located at:

/etc/lightdm/lightdm.conf

This is the primary file that controls how LightDM works, including which greeter will be used, the default desktop environment, autologin settings, and how it handles guests and sessions. It also contains advanced options such as support for XDMCP and VNC.

Main Sections in lightdm.conf

[LightDM]

- run-directory=/run/lightdm: Specifies the path LightDM uses to store runtime data (PID files, sockets).
- log-directory and cache-directory (disabled by default): Control where log and temporary files are stored.

These settings are important for debugging login screen issues.

[Seat:*]

A "Seat" refers to a session associated with a display. The most important settings are:

- greeter-session=lightdm-slick-greeter: Specifies the "greeter" program to be used (in this case, slick-greeter).
- user-session=cinnamon: Defines the default desktop environment after a user logs in (Cinnamon).
- session-wrapper=/etc/lightdm/Xsession: An intermediary script used by LightDM to launch the user's session.

Other settings include:

- autologin-user: Enables automatic login for a specific user.
- greeter-hide-users: Hides the user list.
- allow-guest: Allows a guest account.

This flexibility makes the file customizable for a single workstation or a multi-seat system.

[XDMCPServer]

This section allows for remote connections using the XDMCP protocol. It is disabled by default (enabled=false) for security reasons.

[VNCServer]

This enables remote login via VNC. It is also disabled by default. It can be useful in technical support or classroom environments.

2.6.9.1 slick-greeter: Customizing the Login Screen

The configuration file for slick-greeter is located at:

/etc/lightdm/slick-greeter.conf

This file is responsible for the look and graphical experience of the login screen.

Important Settings in [Greeter]

- background=/usr/share/backgrounds/login.png: The primary background for the login screen.
- theme-name=Arc-Dark: The theme used on the login screen.
- icon-theme-name=Qogir: The default icon theme.
- cursor-theme-name=Qogir and cursor-theme-size=16: The appearance and size of the mouse cursor.
- draw-user-backgrounds=false: Prevents the use of a personal background for each user.
- show-a11y=false and show-power=false: Hides accessibility options and power buttons from the login screen.
- background-color=#000000: An alternative background color (in case the image fails to load).

Summary

- lightdm.conf: This is the central file that defines how LightDM works (sessions, autolog-in, remote protocols).
- slick-greeter.conf: This file is dedicated to the appearance of the login screen (back-ground, theme, icons).

This chapter clearly demonstrates the separation of functionality from appearance:

- Functionality is in lightdm.conf.
- Appearance is in slick-greeter.conf.

2.6.10Network Configuration Files

/etc/resolv.conf

- **Purpose:** Specifies the DNS servers the system uses for name resolution.
- **Typical Content:**
- nameserver 1.1.1.1
- nameserver 8.8.8.8
- **Helwan/Archiso Notes:** In a live environment, this file is usually managed by systemd-resolved or generated by NetworkManager/dhcpcd. Avoid writing static DNS entries if the system uses systemd-resolved. Let resolv.conf be auto-generated or create a symbolic link to /run/systemd/resolve/stub-resolv.conf.

/etc/systemd/network/20-ethernet.network

- **Purpose:** Defines network policies for Ethernet interfaces (static or DHCP).
- **Typical Content:**
- [Match]
- Name=en*
-
- [Network]
- DHCP=yes
- **Notes:** The ISO uses DHCP to simplify network access across various devices. For a static IP during installation, the guide explains how to change DHCP=no and add Address=, Gateway=, and DNS= settings.

/etc/systemd/network/20-wlan.network

- **Purpose:** Rules for Wi-Fi interfaces (wlan*).
- **Typical Content:**
- [Match]
- Name=wlan*
-
- [Network]
- DHCP=yes
- **Notes:** Typically kept simple in live environments, as tools like NetworkManager or iwd handle actual Wi-Fi connections. These .network files ensure wlan interfaces obtain DHCP addresses once connected. Explain the relationship between iwd or wpa_supplicant and this file, if applicable.

/etc/systemd/network/20-wwan.network

- **Purpose:** Settings for cellular (WWAN) communication interfaces.
- **Typical Content:** Similar to wlan but may include IPv6AcceptRA=no or modem-specific configurations.

- **Notes:** Important for devices using cellular data. In live environments, it's usually left to network managers like ModemManager.

/etc/systemd/network.conf.d/ipv6-privacy-extensions.conf

- **Purpose:** Configures IPv6 privacy policy (RFC 4941) by generating temporary addresses to reduce device tracking.
- **Typical Content:**
- [Network]
- PrivacyExtensions=1
- **Notes:** Beneficial for user privacy. In some networks, it might need to be disabled (0) for compatibility reasons. Mention its impact: **improved privacy** versus **difficulty tracking a static interface**.

/etc/systemd/resolved.conf.d/archiso.conf

- **Purpose:** Additional settings for systemd-resolved within the ISO environment.
- **Contextual Typical Content:**
- [Resolve]
- DNS=1.1.1.1
- 8.8.8.8
- FallbackDNS=9.9.9.9
- Cache=yes
- DNSOverTLS=no
- **Helwan Notes:** In the live environment, it's desirable to provide known, stable DNS servers (e.g., Cloudflare/Google) to avoid exceptions during initial user setup. However, after installation, it's recommended to let the user choose or use a reflector/NetworkManager to update the settings.

/etc/ssh/sshd_config.d/10-archiso.conf

- **Purpose:** Custom SSH settings for the ISO/live environment.
- **Examples of Expected Options:**
- # Allow SSH for remote troubleshooting in live environment
- PermitRootLogin=yes
- PasswordAuthentication=yes
- # or maybe a restricted setting: PermitRootLogin=prohibit-password
- **Security Notes:** In live environments, SSH can be enabled for easier remote support. However, it's crucial to always state in the documentation that these settings are **unsuitable for an installed system**. After installation, it's preferable to disable PermitRootLogin or restrict access to key-based authentication only.

Quick Summary (Practical, No Fluff)

- **resolv.conf**: Controls DNS. Let systemd-resolved or NetworkManager handle it in the live environment.
- **systemd/network/20-*.network files**: Ensure automatic DHCP for interfaces in the live environment.
- **ipv6-privacy-extensions.conf**: Manages IPv6 privacy; explain the option and its effects.
- **resolved.conf.d/archiso.conf**: Provides static/secure DNS in the live environment for a reliable experience.
- **sshd_config.d/10-archiso.conf**: Enables SSH specifically for the live environment; warn readers against leaving it enabled on an installed system.

/etc/systemd/system/livecd-alsa-unmuter.service

- **Purpose:** A service designed to unmute ALSA sound devices by default in the live environment. This is because some Linux sound cards open in a muted state, which can confuse users into thinking the sound isn't working.
- **Typical Content:**
- [Unit]
- Description=Unmute ALSA sound devices in Live CD
- After=sound.target
-
- [Service]
- Type=oneshot
- ExecStart=/usr/bin/alsactl init
- ExecStart=/usr/bin/amixer -c 0 set Master unmute
- ExecStart=/usr/bin/amixer -c 0 set PCM unmute
-
- [Install]
- WantedBy=multi-user.target
- **Explanation:**
    - ExecStart=/usr/bin/alsactl init: Initializes ALSA sound settings.
    - amixer set Master/PCM unmute: Unmutes the main audio channels.
    - WantedBy=multi-user.target: The service starts automatically with the system in the live environment.
- **Helwan Notes:** This file is beneficial as it ensures a smooth experience: the system boots up with sound working without manual configuration. After installing to the hard drive, it's preferable to leave ALSA or PulseAudio/PipeWire configuration to the user's preference.

/etc/systemd/system/getty@tty1.service.d/autologin.conf

- **Purpose:** Enables automatic login for the liveuser user on terminal TTY1 when the live system boots.
- **Typical Content:**
- [Service]
- ExecStart=
- ExecStart=-/sbin/agetty --autologin liveuser --noclear %I $TERM
- **Explanation:**
    - ExecStart= (empty line): Clears the original value.
    - agetty --autologin liveuser: Logs liveuser in directly without a password.
    - --noclear: Prevents clearing the TTY screen upon login.
- **Helwan Notes:** A necessary feature for ease of use, especially for beginners. After installation, this setting is disabled, and the normal behavior (user enters a password to log in) is restored.

/etc/systemd/system/display-manager.service

Purpose: A generic service that determines which **Display Manager** will be run on the system. In Helwan Linux → Linked to **LightDM**.

Typical Content:

Ini, TOML

[Unit]

Description=Display Manager

Conflicts=getty@tty1.service

After=systemd-user-sessions.service

getty@tty1.service

[Service]

ExecStart=/usr/bin/lightdm

Restart=always

[Install]

Alias=display-manager.service

WantedBy=graphical.target

Explanation:

- **Conflicts=getty@tty1.service** → Prevents getty from running simultaneously with LightDM.
- **ExecStart=/usr/bin/lightdm** → Here, the choice was **LightDM** (with slick-greeter).
- **WantedBy=graphical.target** → Ensures LightDM starts when entering graphical mode.

Helwan Notes: Linking to **LightDM** adds a direct graphical user experience. After installation, users can change to any other Display Manager (GDM, SDDM…).

Quick Summary:

- **livecd-alsa-unmuter.service** → Sound works in the Live environment without being muted.
- **autologin.conf (part of getty@tty1)** → Direct login for liveuser without a password.
- **display-manager.service** → Specifies the startup of **LightDM** as the default Display Manager.

/etc/systemd/system/choose-mirror.service

Purpose: Automatically selects the fastest **mirrors** when booting into the **Live environment**. This aims to improve package download speeds and system installation efficiency.

Typical Content:

Ini, TOML

[Unit]

Description=Choose the fastest pacman mirror

After=network-online.target

Wants=network-online.target

[Service]

Type=oneshot

ExecStart=/usr/bin/reflector --protocol https --latest 20 --sort rate --save /etc/pacman.d/mirrorlist

[Install]

WantedBy=multi-user.target

Explanation:

- After=network-online.target: The service waits until the network is fully operational before proceeding.
- ExecStart=reflector ...: Utilizes the reflector tool to identify the 20 fastest HTTPS servers, sorting them by connection rate.
- WantedBy=multi-user.target: Ensures this service starts automatically when entering the multi-user mode.

Helwan Notes: This step is **ingenious** as it significantly reduces installation time and provides a smooth experience, even for users geographically distant from the primary Arch Linux servers.

/etc/systemd/system/reflector.service

Purpose: Periodically updates the mirrors based on the settings defined in /etc/xdg/reflector/reflector.conf.

Typical Content:

Ini, TOML

[Unit]

Description=Pacman mirrorlist update

[Service]

Type=oneshot

ExecStart=/usr/bin/reflector --config /etc/xdg/reflector/reflector.conf

Explanation:

- reflector --config: Reads configuration from the specified file and updates the mirrorlist.
- This service can be run manually or integrated with a **timer** for scheduled updates.

Helwan Notes: This ensures the system is not permanently tied to a slow server, adding long-term flexibility and speed.

/etc/systemd/system/reflector.timer

Purpose: Defines the schedule for running the reflector.service.

Typical Content:

Ini, TOML

[Unit]

Description=Run reflector weekly

[Timer]

OnBootSec=10min

OnUnitActiveSec=1w

[Install]

WantedBy=timers.target

Explanation:

- OnBootSec=10min: The initial run occurs 10 minutes after booting.
- OnUnitActiveSec=1w: Subsequent updates will run automatically once every week.
- WantedBy=timers.target: Registers this timer within the systemd timer management system.

Helwan Notes: This is a smart solution that ensures mirror updates happen **without manual intervention**. Users can effectively forget about it, and the system will self-manage.

/etc/systemd/system/systemd-networkd.service

Purpose: Manages network connectivity using systemd-networkd instead of NetworkManager. This is suitable for **lightweight environments or Live systems**.

Typical Content:

Ini, TOML

[Unit]

Description=Network Service

Documentation=man:systemd-networkd.service(8)

ConditionCapability=CAP_NET_ADMIN

After=network-pre.target

Before=network.target

Wants=network.target

[Service]

ExecStart=/usr/lib/systemd/systemd-networkd

Restart=always

[Install]

WantedBy=multi-user.target

/etc/systemd/system/choose-mirror.service

Purpose: Automatically selects the fastest mirrors (mirrors) when the **Live** environment boots. The goal is to improve package download speed and system installation.

Typical Content:

Ini, TOML

[Unit]

Description=Choose the fastest pacman mirror

After=network-online.target

Wants=network-online.target

[Service]

Type=oneshot

ExecStart=/usr/bin/reflector --protocol https --latest 20 --sort rate --save /etc/pacman.d/mirrorlist

[Install]

WantedBy=multi-user.target

Explanation:

- After=network-online.target: The service waits until the network is ready.

- ExecStart=reflector ...: Uses the **Reflector** tool to select the 20 fastest HTTPS servers and sort them by speed.
- WantedBy=multi-user.target: Starts automatically upon entering multi-user mode.

Helwan Notes: This step is brilliant because it saves installation time and makes the experience comfortable, even if the user is far from the primary Arch servers.

/etc/systemd/system/reflector.service

Purpose: Periodically updates the mirrors according to the settings in /etc/xdg/reflector/reflector.conf.

Typical Content:

Ini, TOML

[Unit]

Description=Pacman mirrorlist update

[Service]

Type=oneshot

ExecStart=/usr/bin/reflector --config /etc/xdg/reflector/reflector.conf

Explanation:

- reflector --config: Reads settings from the config file and updates the mirrorlist. This can be run manually or linked to a timer service for periodic updates.

Helwan Notes: This ensures the system isn't stuck with a slow server. It adds flexibility and speed in the long run.

/etc/systemd/system/reflector.timer

Purpose: Schedules the execution of the reflector.service.

Typical Content:

Ini, TOML

[Unit]

Description=Run reflector weekly

[Timer]

OnBootSec=10min

OnUnitActiveSec=1w

[Install]

WantedBy=timers.target

Explanation:

- OnBootSec=10min: First run 10 minutes after boot.
- OnUnitActiveSec=1w: Afterwards, it updates mirrors automatically every week.
- WantedBy=timers.target: Registers itself among systemd's timers.

Helwan Notes: A smart solution that ensures mirrors are updated without manual intervention. The user can forget about it, and the system self-manages.

/etc/systemd/system/systemd-networkd.service

Purpose: Manages the network using systemd-networkd instead of NetworkManager (suitable for lightweight or Live environments).

Typical Content:

Ini, TOML

[Unit]

Description=Network Service

Documentation=man:systemd-networkd.service(8)

ConditionCapability=CAP_NET_ADMIN

After=network-pre.target

Before=network.target

Wants=network.target

[Service]

ExecStart=/usr/lib/systemd/systemd-networkd

Restart=always

[Install]

WantedBy=multi-user.target

Explanation:

- ExecStart: Runs the daemon responsible for network management (interfaces, DHCP, static config...).
- After=network-pre.target and Before=network.target: Ensures the correct order for network preparation before use.
- Restart=always: The service restarts if it fails.

Helwan Notes: systemd-networkd is an excellent choice for a Live ISO because it's lightweight, fast, and doesn't require a graphical interface. After installation, the user can replace it with NetworkManager if they desire a GUI.

/etc/systemd/system/systemd-resolved.service

Purpose: Provides a DNS resolver service integrated with systemd.

Typical Content:

Ini, TOML

[Unit]

Description=Network Name Resolution

Documentation=man:systemd-resolved.service(8)

After=network.target

[Service]

ExecStart=/usr/lib/systemd/systemd-resolved

Restart=always

[Install]

WantedBy=multi-user.target

Explanation:

- Manages files like /etc/resolv.conf.
- Provides support for DNS over TLS (DoT) and DNS caching.
- Integrates with networkd or NetworkManager.

Helwan Notes: A powerful feature because it ensures that domain names are resolved quickly and efficiently, which makes a difference in the installation and package download experience.

Quick Summary (This part relates to systemd):

- choose-mirror.service: Selects the fastest mirrors on boot.
- reflector.service + reflector.timer: Updates mirrors periodically (weekly).
- systemd-networkd.service: Manages networks in a lightweight manner.
- systemd-resolved.service: Resolves and caches domain names.

/etc/systemd/system/display-manager.service

Purpose: A generic service that defines which **Display Manager** will be launched on the system. In Helwan Linux, this is linked to **LightDM**.

Typical Content:

Ini, TOML

[Unit]

Description=Display Manager

Conflicts=getty@tty1.service

After=systemd-user-sessions.service

getty@tty1.service

[Service]

ExecStart=/usr/bin/lightdm

Restart=always

[Install]

Alias=display-manager.service

WantedBy=graphical.target

Explanation:

- Conflicts=getty@tty1.service: Prevents getty from running simultaneously with LightDM.
- ExecStart=/usr/bin/lightdm: The choice here is **LightDM** (with slick-greeter).
- WantedBy=graphical.target: Ensures LightDM starts when the system enters graphical mode.

Helwan Notes: Linking to LightDM provides a direct graphical user experience. After installation, users can switch to any other Display Manager (GDM, SDDM, etc.).

Quick Summary:

- livecd-alsa-unmuter.service: Ensures audio is working in the Live environment without being muted.
- autologin.conf (part of getty@tty1): Enables direct login for liveuser without a password.
- display-manager.service: Defines LightDM as the default display manager.

/etc/systemd/system/choose-mirror.service

Purpose: Automatically selects the fastest **mirrors** when the **Live** environment boots. The goal is to optimize package download speed and system installation.

Typical Content:

Ini, TOML

[Unit]

Description=Choose the fastest pacman mirror

After=network-online.target

Wants=network-online.target

[Service]

Type=oneshot

ExecStart=/usr/bin/reflector --protocol https --latest 20 --sort rate --save /etc/pacman.d/mirrorlist

[Install]

WantedBy=multi-user.target

Explanation:

- After=network-online.target: The service waits until the network is ready.
- ExecStart=reflector ...: Uses the Reflector tool to select the 20 fastest HTTPS servers and sort them by rate.
- WantedBy=multi-user.target: Automatically starts upon entering the multi-user mode.

Helwan Notes: This step is brilliant as it significantly reduces installation time and makes the experience smooth, even for users geographically distant from the primary Arch servers.

/etc/systemd/system/reflector.service

Purpose: Periodically updates the mirrors based on the settings in /etc/xdg/reflector/reflector.conf.

Typical Content:

Ini, TOML

[Unit]

Description=Pacman mirrorlist update

[Service]

Type=oneshot

ExecStart=/usr/bin/reflector --config /etc/xdg/reflector/reflector.conf

Explanation:

- reflector --config: Reads configuration from the specified file and updates the mirrorlist. This can be run manually or linked to a timer service for periodic updates.

Helwan Notes: This ensures the system isn't stuck with a slow server, adding long-term flexibility and speed.

/etc/systemd/system/reflector.timer

Purpose: Defines the schedule for running the reflector.service.

Typical Content:

Ini, TOML

[Unit]

Description=Run reflector weekly

[Timer]

OnBootSec=10min

OnUnitActiveSec=1w

[Install]

WantedBy=timers.target

Explanation:

- OnBootSec=10min: The first run occurs 10 minutes after boot.
- OnUnitActiveSec=1w: Subsequently, mirrors are updated automatically every week.
- WantedBy=timers.target: Registers with systemd's timers.

Helwan Notes: A smart solution that ensures mirror updates without manual intervention. Users can effectively forget about it, and the system manages itself.

/etc/systemd/system/systemd-networkd.service

Purpose: Manages networking using systemd-networkd instead of NetworkManager. This is suitable for lightweight environments or Live systems.

Typical Content:

Ini, TOML

[Unit]

Description=Network Service

Documentation=man:systemd-networkd.service(8)

ConditionCapability=CAP_NET_ADMIN

After=network-pre.target

Before=network.target

Wants=network.target

[Service]

ExecStart=/usr/lib/systemd/systemd-networkd

Restart=always

[Install]

WantedBy=multi-user.target

Explanation:

- ExecStart: Starts the daemon responsible for managing network interfaces, DHCP, and static configurations.
- After=network-pre.target and Before=network.target: Ensures correct ordering for network preparation before it's used.
- Restart=always: The service restarts if it fails.

Helwan Notes: systemd-networkd is an excellent choice for the Live ISO because it's lightweight, fast, and doesn't require a graphical interface. After installation, users can replace it with NetworkManager if they prefer a GUI.

/etc/systemd/system/systemd-resolved.service

Purpose: Provides an integrated DNS resolver service with systemd.

Typical Content:

Ini, TOML

[Unit]

Description=Network Name Resolution

Documentation=man:systemd-resolved.service(8)

After=network.target

[Service]

ExecStart=/usr/lib/systemd/systemd-resolved

Restart=always

[Install]

WantedBy=multi-user.target

Explanation:

- Manages files like /etc/resolv.conf.
- Offers support for DNS over TLS (DoT) and DNS caching.
- Integrates with networkd or NetworkManager.

Helwan Notes: A powerful feature that ensures domain names are resolved quickly and efficiently, which significantly impacts the installation and package download experience.

/etc/systemd/system/systemd-timesyncd.service

Purpose: Synchronizes the clock and date with NTP (Network Time Protocol) servers. This is crucial for the correct functioning of package GPG signatures and to prevent system file inconsistencies due to time differences.

Typical Content:

Ini, TOML

[Unit]

Description=Network Time Synchronization

Documentation=man:systemd-timesyncd.service(8)

ConditionCapability=CAP_SYS_TIME

ConditionVirtualization=!container

After=network.target

systemd-networkd.service

Wants=network-online.target

[Service]

ExecStart=/usr/lib/systemd/systemd-timesyncd

Restart=always

[Install]

WantedBy=multi-user.target

Alias=dbus-org.freedesktop.timesync1.service

Explanation:

- After=network.target: Waits for the network to be ready.
- ExecStart: Starts the daemon responsible for fetching the time from the internet.
- Alias=dbus-org.freedesktop.timesync1.service: Ensures integration with D-Bus, so any program requesting a unified time will find it.

Helwan Notes: This is extremely important because without accurate time, pacman will report "invalid signature" when downloading any package. It instills confidence that users won't encounter installation issues.

/etc/systemd/system/dhcpcd.service

Purpose: A DHCP client service to automatically obtain an IP address from the router.

Typical Content:

Ini, TOML

[Unit]

Description=dhcpcd on all interfaces

Wants=network.target

Before=network.target

[Service]

Type=forking

ExecStart=/usr/bin/dhcpcd -q -b

PIDFile=/run/dhcpcd.pid

Restart=always

[Install]

WantedBy=multi-user.target

Explanation:

- ExecStart=/usr/bin/dhcpcd -q -b: Runs the DHCP client in the background.
- PIDFile=/run/dhcpcd.pid: Saves the PID so systemd can manage the service.
- Before=network.target: Must run before the network is considered "ready."

Helwan Notes: In the Live ISO, a reliable method is needed to connect to the internet quickly. dhcpcd achieves this efficiently without complex configurations.

/etc/systemd/system/getty@tty1.service

Purpose: Provides a text-based login prompt (**tty1**) for users in case graphical login fails or when the system boots into CLI mode.

Typical Content:

Ini, TOML

[Unit]

Description=Getty on tty1

After=systemd-user-sessions.service

plymouth-quit-wait.service

Conflicts=rescue.service

rescue.target

[Service]

ExecStart=-/sbin/agetty -o '-p -- \\u' --noclear tty1 linux

Restart=always

RestartSec=0

[Install]

WantedBy=graphical.target

Explanation:

- agetty: The program responsible for presenting the login prompt on the tty.
- --noclear: Prevents clearing the tty1 screen after boot.
- Restart=always: If the session is closed, it will automatically open a new one.

Helwan Notes: This is an excellent backup if LightDM fails. It can also be relied upon entirely in server environments or on low-spec machines.

/etc/systemd/system/graphical.target

Purpose: This is a **target**, not a service. It represents the system's operational state in graphical mode (GUI).

Typical Content:

Ini, TOML

[Unit]

Description=Graphical Interface

Requires=multi-user.target

After=multi-user.target

AllowIsolate=yes

Explanation:

- Requires=multi-user.target: The user session system (CLI) must start before the GUI.
- After=multi-user.target: Ensures correct ordering.
- AllowIsolate=yes: Allows manual switching to graphical mode using: systemctl isolate graphical.target

---

Summary (systemd Component):

- choose-mirror.service: Selects the fastest mirrors on boot.
- reflector.service + reflector.timer: Updates mirrors periodically (weekly).
- systemd-networkd.service: Manages networks in a lightweight manner.
- systemd-resolved.service: Resolves domain names and caches them.
- systemd-timesyncd.service: Synchronizes system time with NTP servers.
- dhcpcd.service: Obtains an IP address automatically via DHCP.
- getty@tty1.service: Provides a text-based login prompt as a fallback.
- graphical.target: Represents the graphical user interface mode

2.6.11Function of the /etc/skel/ Directory

The /etc/skel/ directory, often referred to as the "skeleton directory," serves as a template for new user accounts. When a new user is created using the useradd command with the -m option (which ensures the home directory is created), the contents of /etc/skel/ are automatically copied into the new user's home directory (/home/username/). This means any file or folder present in /etc/skel/ will be replicated for each new user.

Primary Use Cases:

- **Customizing the Default User Environment:** This includes essential shell configuration files like .bashrc, .zshrc, .profile, and .bash_profile. These files allow for pre-configuration of shell settings, aliases, command-line colors, and other user preferences.
- **Setting Up the Desktop Environment:** Developers can pre-configure desktop environment settings by placing files within the .config/ directory. This might include default settings for text editors, themes, wallpapers, or template files that new users can start with.
- **Providing Instructions or Documentation for New Users:** The skeleton directory can be used to include introductory files such as a README or WELCOME file in the user's home directory. It can also contain text files with getting started instructions or relevant links.

Application in Arch-Based Distributions (like Helwan Linux):

In distributions derived from Arch Linux, such as Helwan Linux, the /etc/skel/ directory might initially be empty or contain only very basic files (e.g., a default .bashrc). However, as a distribution developer, you can leverage this directory to:

- **Pre-configure Shell or Desktop Environment Settings:** Add ready-to-use configurations for the shell or a chosen desktop environment.
- **Establish a Consistent Initial User Experience:** Ensure a unified look and feel for all new users.
- **Provide a Pre-configured Environment:** Allow every new user to begin with a pre-set environment rather than a completely blank slate.

Suggested Text for Documentation:

2.x /etc/skel — The Initial Template for New Users

The /etc/skel/ directory acts as the **default template** that is automatically copied to the home directory of every new user created using useradd -m. Any file or directory within /etc/skel/ will be replicated in /home/username/.

Configuration files such as .bashrc and .profile are used to **customize the shell environment**. Developers can add custom settings, like files in .config/ to prepare the desktop environment or

text editors. This directory can also be used to **include welcome files or instructions** for the new user.

In distributions like Helwan Linux, this directory is a **key point for standardizing the default user experience**, ensuring all users start with pre-configured settings instead of a default raw environment.

2.7 In the Lab

In this section of the book, we'll practically learn how to customize a distribution in a cohesive and sound manner. We'll start from scratch to leverage what we've previously discussed in the earlier sections.

Whatever your inclinations or leanings, don't be shy about imprinting your version of the distribution with those leanings. Let the distribution loudly announce its identity. This is what will make your distribution unique. Don't try to please users at the expense of the distribution's direction; attempting to please everyone will lead you to lose everyone and turn your project into a clumsy imitation, lacking identity, or just another copy of the Mona Lisa, or loudly proclaim to the world that it's a clown's attempt to imitate a famous actor.

Careful Package Bundling

Package bundling is what gives a distribution its flavor and taste, not just its look and feel. Therefore, don't burn out your package management easily and turn it into a source of failure instead of success. Choose packages carefully before adding them to the distribution. Ask yourself: Why this package? Will it reinforce the distribution's identity? Will it serve the target user? Can I program a similar package that spares the user the pain of using this one? And so on, don't be hasty in stuffing packages into the distribution; every added package must be well-thought-out and serve the distribution's purpose.

The profiledef.sh File – Your Loyal Friend

Another important card that stands out in the game is the profiledef.sh file. This file's role isn't limited to just defining the distribution's name; it also deals with file permissions (file_permissions).

You can consider it your loyal friend, who makes it easy for you to accomplish tasks without bearing the burden of everything yourself. This file works reliably in the background, flawlessly completing powerful tasks for you, like a diligent employee who never tires.

Are you ready to enter the lab to meticulously dissect the distribution's core?

Installing Archiso

Archiso is the official tool for creating bootable ISO images for Arch-based distributions. To install it on Arch Linux, run:

Bash

sudo pacman -S archiso

This is sufficient to set up your working environment for ISO creation. It's recommended to also install auxiliary tools like base-devel, vim, and git if you plan on extensive customization.

---

2. Understanding Profile Differences

After installing archiso, you'll find the available profiles located at:

/usr/share/archiso/configs/

Key Profiles:

Profile	Description
releng/	The official release profile. It represents the current Arch Linux stable version and is used to generate an ISO identical to the base Arch distribution.
baseline/	A stripped-down version of Arch containing only the essential packages to minimize the ISO size.
next/	An experimental/development profile that includes new tools and tests not yet officially adopted.

Practical Notes:

- Most new distribution projects begin with the releng/ profile due to its stability and assurance of official functionality.
- The baseline/ profile is useful if you require a very small, fully customizable ISO.

3. Copying the releng Profile

To start customizing your own distribution, copy the releng/ profile to your working directory:

Bash

```
mkdir ~/my-archiso

cp -r /usr/share/archiso/configs/releng/ ~/my-archiso/

cd ~/my-archiso/releng/
```

You can now:

- Modify Packages: Edit the packages.x86_64 file to add or remove packages.
- Adjust Configuration Files: Modify files within airootfs/ and profiledef.sh.
- Add Custom Scripts or Tools: Integrate any specific scripts or utilities for your distribution.

After making your modifications, you can build your custom ISO:

Bash

```
sudo mkarchiso -v .
```

The -v flag enables verbose mode, displaying detailed steps during the build process.

Archiso Configuration Structure:

Archiso Configs

(/usr/share/archiso/configs/)

```
│
├── releng/       ← Official stable release (ideal starting point for ISO customization)
│   ├── packages.x86_64    ← Core distribution packages
│   ├── airootfs/          ← The virtual root filesystem for the ISO
│   ├── profiledef.sh      ← Defines file permissions and settings
│   └── …                  ← Other configuration files
```

│

├── baseline/ ← Very minimal version with few packages, suitable for small-scale projects

│

└── next/ ← Experimental/development version with unstable tools

---

Practical Customization Steps:

1. Copy the Official Profile:

Bash

mkdir ~/my-archiso

cp -r /usr/share/archiso/configs/releng/ ~/my-archiso/

cd ~/my-archiso/releng/

2. Edit Packages:
    - packages.x86_64: Add or remove packages to align with your distribution's desired identity.
3. Modify Root Filesystem and Configuration:
    - airootfs/: Add your custom files or applications here.
    - profiledef.sh: Adjust permissions and specific settings for your distribution.
4. Build Your Custom ISO:

Bash

sudo mkarchiso -v .

2.7.1 Modifying the `profiledef.sh` File

We will divide this file into three sections to facilitate a smooth and straightforward workflow.

Section 1: ISO and Build Definitions

Bash

```
#!/usr/bin/env bash
# Suppress a specific warning from the ShellCheck tool (SC2034 indicates a
defined but unused variable).
# shellcheck disable=SC2034

# **ISO Name:**
# Defines the final name of the ISO file. This is the name the user will see
and interact with when downloading or distributing it.
iso_name="Helwan-Linux-stable"

# **ISO Label:**
# The label written to the ISO when it's burned or when the CD/USB is booted.
iso_label="Helwan-Linux-stable-v1.1"

# **Publisher Information:**
# Appears in the ISO's metadata.
iso_publisher="helwanlinux <helwanlinux@gmail.com>"

# **Application Description:**
# Describes the purpose of the ISO. In this case, it's for a live/rescue en-
vironment.
iso_application="Helwan Linux Live/Rescue DVD"

# **ISO Version:**
# An internal version number for developers; not typically displayed to the
end-user.
iso_version="v1.1"

# **Installation Directory:**
# Specifies the name of the directory within the ISO that contains the core
installation files.
# For Arch-based systems, this is conventionally named 'arch' because boot-
loaders (like syslinux or grub) are configured to look for it.
install_dir="arch"

# **Build Modes:**
# Specifies the type of output the `mkarchiso` tool will generate.
# The value 'iso' indicates that the final output will be a bootable ISO
file, typical for Linux distributions.
# Other modes, such as 'netboot' or 'bootstrap', can be added for specialized
use cases like network booting or creating a base system for further builds,
though they are less common.

# Example:
# If you set:
# buildmodes=('iso' 'bootstrap')
# The `mkarchiso` tool will generate two outputs:
```

```
# 1. A bootable ISO file.
# 2. A bootstrap directory containing essential system files that can serve
as a base for other builds.
buildmodes=('iso')
```

This section defines critical metadata for the ISO image and specifies how the build process should be executed.

This section of the `profiledef.sh` file defines several key parameters for building an Arch Linux-based ISO.

Target Architecture

- **`arch="x86_64"`**: This line specifies the **target architecture** for which the ISO will be built. The value `x86_64` indicates that the ISO will be compatible with **64-bit systems**.

Pacman Configuration File

- **`pacman_conf="pacman.conf"`**: This parameter points to the **Pacman configuration file** that will be used during the build process. Typically, this would be a local `pacman.conf` file, allowing for customization of repositories and other Pacman options.

AIROOTFS Image Type

- **`airootfs_image_type="squashfs"`**: This setting defines the **image type** for the system's root file system (`airootfs`). Using `squashfs` means the files will be compressed into a **read-only SquashFS format**. This is beneficial for achieving a **smaller size** and faster loading times during runtime.

AIROOTFS Image Tool Options

- **`airootfs_image_tool_options=('-comp' 'xz' '-Xbcj' 'x86' '-b' '1M' '-Xdict-size' '1M')`**: These are **additional options** passed to the SquashFS compression tool when creating the `airootfs` image.
  - `-comp xz`: Utilizes the **`xz` compression algorithm**.
  - `-Xbcj x86`: Applies **x86-specific branch/jump (BCJ) filtering** for improved compression.
  - `-b 1M`: Sets the **block size to 1MB**, which can enhance efficiency.
  - `-Xdict-size 1M`: Sets the **dictionary size for compression to 1MB**.

Bootstrap Tarball Compression

- **`bootstrap_tarball_compression=('zstd' '-c' '-T0' '--auto-threads=logical' '--long' '-19')`**: This specifies the **compression method** for the bootstrap archive, which contains the core system environment.

- `zstd`: Employs the **Zstandard (zstd) compression algorithm**, known for being faster and more efficient than `xz`.
- `-c`: Writes the **compressed data to standard output (stdout)**.
- `-T0`: Instructs `zstd` to **use all available processor cores**.
- `--auto-threads=logical`: Automatically determines the **number of threads** based on the number of logical cores.
- `--long`: Activates the **"long" mode** for maximum compression efficiency.
- `-19`: Sets the compression level to **19**, the highest available level for maximum compression.

Section 3: File Permissions (file_permissions) in `profiledef.sh`

This section of the `profiledef.sh` file is crucial for defining the **permissions** of essential files and directories that require special attention during the distribution build process. The primary goal is to ensure that sensitive files are not accessible to regular users, while simultaneously allowing necessary execution or read access for the system and users during live sessions or installation.

Understanding File Permissions

File permissions in Unix-like systems are represented by a three-digit octal number, where each digit corresponds to permissions for the **owner**, the **group**, and **others**, respectively. The digits represent the sum of specific permissions:

- **4**: Read (r)
- **2**: Write (w)
- **1**: Execute (x)

For example, `750` means:

- **Owner:** 7 (4+2+1 = read, write, execute)
- **Group:** 5 (4+1 = read, execute)
- **Others:** 0 (no permissions)

The format used in `profiledef.sh` is generally `"owner:group:permissions"`, where `0:0` typically signifies ownership by the **root** user and the **root** group.

Defining Permissions for Key Files and Directories

Here's a breakdown of the specific permissions defined in this section:

- **`/etc/shadow`**: This file stores encrypted user passwords. It must only be readable by the **root** user.
    - `["/etc/shadow"]="0:0:400"`: This sets ownership to **root**: **root**, with **read-only** permissions for **root** (400).
- **`/root`**: This is the personal directory for the **root** user.
    - `["/root"]="0:0:750"`: Ownership is **root**: **root**. The **root** user has full permissions (750: read, write, execute), the group can enter and read (750: read, execute), and others have no permissions.
- **`/root/.automated_script.sh`**: This is a script that runs automatically to streamline system setup.
    - `["/root/.automated_script.sh"]="0:0:755"`: Ownership is **root**: **root**. The **root** user has full permissions, and other users can read and execute the script.
- **`/root/.gnupg`**: This directory stores private encryption keys and needs to be highly protected.
    - `["/root/.gnupg"]="0:0:700"`: Ownership is **root**: **root**. Only the **root** user has full permissions (700); no other access is allowed.
- **`/usr/local/bin/choose-mirror`**: This script helps the system select the best server for updates.
    - `["/usr/local/bin/choose-mirror"]="0:0:755"`: Ownership is **root**: **root**. It's executable by all users.
- **`/usr/local/bin/Installation_guide`**: This script displays installation steps to users.
    - `["/usr/local/bin/Installation_guide"]="0:0:755"`: Ownership is **root**: **root**. It's executable by all users, but only **root** can modify it.
- **`/usr/local/bin/livecd-sound`**: This handles audio playback in the Live environment.
    - `["/usr/local/bin/livecd-sound"]="0:0:755"`: Ownership is **root**: **root**. It's executable by all users.
- **`/etc/polkit-1/rules.d`**: This directory stores rules for controlling user permissions.
    - `["/etc/polkit-1/rules.d"]="0:0:750"`: Ownership is **root**: **root**. Only the system administrator and the group can access it; others have no permissions.
- **`/etc/sudoers.d`**: This directory contains additional configurations for the `sudo` command.
    - `["/etc/sudoers.d"]="0:0:750"`: Ownership is **root**: **root**. Access is restricted to **root** and the group.

Additional Examples for Customization

You can add entries for other files and directories to further secure your distribution:

- **`/etc/ssh/sshd_config`**: To secure SSH configurations.
    - `["/etc/ssh/sshd_config"]="0:0:600"`: Ownership **root**: **root**, read/write only for **root**.
- **`/etc/fstab`**: The file system table, which mounts disks. It should be readable by all but modifiable only by **root**.

    - `["/etc/fstab"]="0:0:644"`: Ownership **root**: **root**, read for all, write only for **root**.
- **`/var/log`**: To protect system logs from tampering or snooping.
    - `["/var/log"]="0:0:750"`: Ownership **root**: **root**, full permissions for **root**, read and execute for the group, no permissions for others.
- **`/boot/grub/grub.cfg`**: The bootloader configuration file, which needs strong security.
    - `["/boot/grub/grub.cfg"]="0:0:600"`: Ownership **root**: **root**, read/write only for **root**.

This detailed configuration of file permissions is essential for building a secure and robust custom Linux distribution.

With this, we've completed our walkthrough of the **`profiledef.sh`** file—from defining ISO metadata, setting the architecture and compression options, to configuring critical file permissions. These configurations form the backbone of any Arch-based distribution, ensuring that the resulting ISO is stable, secure, and ready for practical use. In the upcoming sections, we'll move from preparation to hands-on implementation, shaping the distribution step by step.

Section 2.7.2: Adding Packages in `packages.x86_64`

Initiating Work: Defining Distribution Flavor and Technical Identity

The crucial step now is to define your distribution's unique flavor and technical identity. This is the hook that will attract your target users. When you open the `packages.x86_64` file, you'll find that Arch Linux pre-populates it with essential packages released monthly. However, it notably omits graphical installers, desktop environments, or even your preferred browser. This is where your role and expertise come into play, making your distribution declare, "This is my genetic code from within."

Note: Adding a `#` symbol at the beginning of a line instructs the distribution build process to ignore that line and its contents entirely.

Bash

```
#calamares-packages
calamares
hel-calamares-config
os-prober
ckbcomp
mkinitcpio-openswap
```

Key Package Additions

- **`calamares`**: This is the renowned graphical installer used by hundreds of distributions. Including it makes the installation process of your distribution user-friendly.
- **`hel-calamares-config`**: These are the customization files specific to your distribution (Helwan). They allow you to modify the appearance and behavior of the graphical installer to align with your distribution's identity.
- **`os-prober`**: An essential package that helps detect other operating systems already installed on the system, facilitating dual-boot installations.

Additional Customizations

- **`ckbcomp`**: This utility is used for customizing keyboard layouts and supporting specific languages or unique key mappings.
- **`mkinitcpio-openswap`**: This package adds support for unlocking encrypted swap partitions during the boot process, thereby enhancing system security.

GUI Packages

```
#GUI-packages##

xorg
xorg-apps
xf86-input-libinput
xdg-user-dirs
```

```
lightdm
lightdm-slick-greeter
base-devel
pacman-contrib
bash-completion
zenity
```

This section encompasses the **essential packages** required to run a **complete graphical user interface (GUI)** on your distribution. Without these components, the system would remain in **command-line interface (CLI) mode** only. The following packages represent the **backbone of the graphical part** and complete the distribution's technical identity.

1. `xorg`

The **X.Org Server** is the fundamental component for running any graphical interface on Linux. Its function is to **manage the display**, rendering windows and graphical applications. Without it, no desktop environment can be launched.

2. `xorg-apps`

A collection of **utility tools** specific to Xorg, such as:

- `xrandr`: For **changing screen resolution**.
- `xset`: For **input and output settings**.
- `xhost`: For **controlling access to the X server**.

Including these provides essential tools for **maintaining and configuring the graphical environment**.

3. `xf86-input-libinput`

This package contains the **input drivers** for mice and keyboards. It relies on the `libinput` library and is considered the **default standard** in most modern distributions. Without it, input devices might not function correctly within the graphical interface.

4. `xdg-user-dirs`

This utility creates **standard user directories** like `Documents`, `Downloads`, and `Pictures`. It offers an **organized and professional experience** for new users and ensures that various applications consistently recognize these directories.

5. `lightdm`

The **graphical login manager** (Display Manager). This is the program that appears on system startup, allowing users to enter their username and password to access their desktop environment.

6. `lightdm-slick-greeter`

The **welcome interface** for LightDM. It adds a modern and elegant login screen (Greeter), enhancing the user experience and giving the distribution a professional visual touch right from the start.

7. `base-devel`

A **core set of development tools** such as `gcc`, `make`, and `autoconf`. Its presence is crucial for any user intending to install or build software from the **Arch User Repository (AUR)** or from source. Its absence hinders many advanced customization options.

8. `pacman-contrib`

A set of **additional utilities for `pacman`**, including:

- `checkupdates`: For **checking available updates**.
- `paccache`: For **removing old packages from the cache**.

These tools make system management more flexible and efficient.

9. `bash-completion`

The **auto-completion feature for commands** in the terminal (Tab Completion). It provides users with speed and convenience when typing long or complex commands, making the CLI experience more user-friendly.

10. `zenity`

A tool that allows the **creation of graphical dialog windows** from within Bash scripts. For example, it can be used to display an error message or a file selection window during script execution. It's highly useful for providing simple graphical interaction within command-line tools.

 Summary:

Collectively, these packages provide the **minimum requirements for running a professional graphical interface** on your distribution. From launching the graphical server (`xorg`) to the login screen (`lightdm-slick-greeter`), with full support for additional tools that simplify life for both users and developers.

Section: Network & Bluetooth Packages

This section focuses on providing **network connectivity** (wired/wireless) and **Bluetooth support**, as well as improving audio and network management in your distribution. Without these

components, users would suffer from the absence of the simplest daily functions, such as connecting to the internet or using Bluetooth headphones.

Network & Bluetooth Core Components:

#Network & Bluetooth

networkmanager

network-manager-applet

bluez

bluez-utils

pipewire-pulse

reflector

1. **`networkmanager`** The **primary network management tool** in most modern Linux distributions.
    - It provides a powerful utility for managing internet connections (Wi-Fi, Ethernet, VPN).
    - It supports automatic connection startup on system boot. Without it, users would have to rely on complex manual configurations like `ip` or `wpa_supplicant`.
2. **`network-manager-applet`** The **graphical front-end for NetworkManager**.
    - It appears as an icon in the taskbar/system tray.
    - It allows users to select available networks, enter passwords, and manage VPNs easily. Its presence transforms the connection experience into something simple and practical instead of typing lengthy commands.
3. **`bluez`** The **core Bluetooth Protocol Stack** for supporting Bluetooth devices on Linux.
    - Without it, the system will not recognize any Bluetooth device.
    - It enables pairing with headphones, keyboards, controllers, etc.
4. **`bluez-utils`** A set of **additional command-line utilities for managing Bluetooth**.
    - Example: `bluetoothctl` - a tool for managing pairing and connections (connect/disconnect).
    - This is crucial for advanced users or for troubleshooting when graphical interfaces fail.
5. **`pipewire-pulse`** A **compatibility layer between PipeWire** (the modern media server) and **PulseAudio**.
    - It makes older applications relying on PulseAudio work seamlessly with PipeWire.
    - It provides stable audio, supports Bluetooth, and operates with higher efficiency compared to traditional PulseAudio.

6. `reflector` A tool for managing **Arch Linux repository mirrors**.
    - It fetches the latest list of mirrors from the official Arch website.
    - It can sort mirrors by speed or last update, ensuring that package downloads are fast and efficient.
    - **Practical Example:**

      Bash

      ```
      reflector --country "Egypt" --age 12 --sort rate --save
      /etc/pacman.d/mirrorlist
      ```

      This command selects the fastest mirrors in Egypt within the last 12 hours.

Summary:

These packages provide the **fundamental infrastructure** for internet connectivity and Bluetooth management, with audio enhancements via PipeWire and ensuring fast update downloads through Reflector. In essence, without this section, the distribution would lack the most critical elements for daily use.

Section: Desktop Environment Packages

##desktop-environment

cinnamon

nemo-fileroller

evince

gedit

xfce4-terminal

gnome-calculator

eog

helfetch

gnome-keyring

This section provides the essential desktop environment and its core applications. While the **GUI Packages** section established the graphical base of the system, these packages define the actual **user experience**. They form the desktop, file management, document handling, and the small utilities that make the distribution functional and friendly.

Core Packages

1. **cinnamon**
    - A modern desktop environment developed by the Linux Mint team.
    - Provides a Windows-like interface that is user-friendly, elegant, and customizable.
    - Includes built-in settings tools, panels, and applets.
    - Chosen for being lightweight yet feature-rich, making it ideal for both new and advanced users.
2. **nemo-fileroller**
    - **Nemo** is the default file manager for Cinnamon.
    - The `fileroller` integration allows direct archive management (extracting/compressing files) from Nemo's right-click menu.
    - Provides a smooth and intuitive file handling experience.
3. **evince**
    - A lightweight document viewer, primarily for PDF and PostScript files.
    - Essential for reading books, documentation, or digital manuals.
    - Known for being simple yet highly efficient.
4. **gedit**
    - The default GNOME text editor, chosen here for its simplicity and stability.
    - Perfect for editing configuration files, writing scripts, or general text editing.
    - Provides syntax highlighting for many programming languages.
5. **xfce4-terminal**
    - A lightweight and fast terminal emulator.
    - Chosen for being resource-friendly while still supporting tabs, color schemes, and custom profiles.
    - Ensures advanced users can work comfortably in CLI mode from within the desktop environment.
6. **gnome-calculator**
    - A simple yet powerful calculator application.
    - Supports basic, scientific, and programmer modes.
    - Provides quick access to calculations without requiring external tools.
7. **eog (Eye of GNOME)**
    - A fast and lightweight image viewer.
    - Supports common image formats (JPEG, PNG, GIF).
    - Includes simple editing features like rotate and zoom, making it perfect for everyday image viewing.
8. **helfetch**
    - A custom system information tool (similar to *neofetch*).
    - Displays system details (kernel, packages, RAM, desktop environment, etc.) in a stylish format.

    - Reinforces the distribution’s branding and identity.
9. **gnome-keyring**
    - A password and secret manager that integrates with the desktop.
    - Stores Wi-Fi passwords, SSH keys, and other credentials securely.
    - Essential for seamless authentication and a polished user experience.

 Summary:
These packages transform the system from a bare graphical shell into a **complete desktop environment**. From Cinnamon’s user-friendly interface, Nemo’s file management, Evince’s document handling, to the utility apps like Calculator and Eye of GNOME — every component contributes to a practical, daily-use distribution. Together, they shape the visual and functional identity of your distro.

2.7.3 Utilities & Repositories

After covering **pacman.conf** in a previous section—where we explained how to configure the official repositories—and after working with the core packages in **bootstrap_packages.x86_64**, it is worth briefly exploring the additional utilities and repositories you may want to include in your distribution.

Repositories

By default, Arch Linux relies on three main repositories:

- **core**: Contains the essential system packages.
- **extra**: Includes desktop applications and common utilities.
- **community**: Provides community-maintained software.
- **multilib** (optional): Supports running 32-bit applications on 64-bit systems.

In addition, you can create and add your own custom repository (e.g., *hel-repo*) to host your distribution's unique packages, branding, and tools.

Utilities

Some utilities can enhance the user experience and provide a more complete system, such as:

- **git**: For version control and retrieving packages from the AUR.
- **wget** or **curl**: For downloading files from the internet via the terminal.
- **nano** or **vim**: Essential text editors for file editing.
- **htop**: An interactive process and resource monitor.
- **man-db** and **man-pages**: Provide command documentation and manual pages.

Summary

Adding these utilities and repositories gives your distribution a strong and flexible foundation. However, since we have already covered the essentials in earlier sections, we will now move on in section **2.7.4** to focus on the **desktop environment and main applications**, where the visual and practical identity of the distribution begins to take shape.

2.7.4 Exploring the `/usr` Directory

Having selected the essential packages, configured permissions, and set up the installation environment, it's time to delve into the backbone of your distribution: the **`/usr` directory**. This directory is the primary home for most applications, libraries, and tools that ensure the system runs smoothly after boot.

In essence, the `/usr` directory contains everything a user needs to run programs and graphical environments, reducing the need to access the root directory or core system files. A careful understanding and management of this directory give you full control to:

- **Customize Applications:** Choose the packages that reflect your distribution's identity and meet user needs.
- **Manage Libraries:** Ensure that libraries and tools are compatible with the additional packages you plan to include.
- **Enhance User Experience:** Improve system efficiency, whether in a desktop environment or via the command line.

In the upcoming sections, we will analyze the key subdirectories within `/usr`, focusing on what should be included and modified to create a robust, secure, and user-friendly distribution.

2.7.4.1 Exploration: The `/usr/share/grub/themes/helwan` Directory

Now that we've grasped the structure and significance of the `/usr` directory within your distribution, it's time to dive into a visually impactful component: your distribution's **GRUB themes**.

This directory houses the files that define the appearance of the **boot menu** when the system starts. Designing this theme isn't just about aesthetics; it significantly reflects your distribution's identity, offering users a first impression of the system's quality from the very first moment.

Understanding the contents of this directory and how to modify them will empower you to:

- **Customize the boot menu's graphical interface** to align with the identity of Helwan Linux.
- **Control GRUB's colors, logos, and fonts.**
- **Enhance the user experience** from the very first second they interact with the system.

In the upcoming sections, we will explain the core files within this directory, providing practical examples for implementing a fully custom theme.

Explaining Key Codes in `theme.txt`

This file allows for extensive customization of the GRUB bootloader's appearance. Here's a breakdown of the most important codes and their functions:

1. Main Options

- `title-text: ""` This defines the title text that appears at the top of the boot interface. Here, it's left empty, meaning no title is displayed.
- `desktop-image: "background.png"` This specifies the image to be used as the background for the boot menu. You can replace `"background.png"` with any other image to reflect your distribution's identity.
- `desktop-color: "#000000"` The color to be used as a background if the image isn't displayed. In this case, it's black.
- `terminal-font: "Terminus Regular 18"` The font type and size used for the Terminal window within GRUB.
- `terminal-box: "terminal_box_*.png"` The shape or image of the frame that appears around the Terminal interface.
- `terminal-left, terminal-top` These define the position of the frame on the screen, specified as a percentage from the left and top edges, respectively.
- `terminal-width, terminal-height` These set the width and height of the Terminal window, also as a percentage of the screen size.
- `terminal-border: "0"` The thickness of the border around the Terminal window. A value of `0` indicates no border.

2. Boot Menu

- `+ boot_menu { ... }` This section defines the design of the boot menu itself:
  - `left, top`: The position of the menu on the screen relative to width and height.
  - `width, height`: The size of the menu on the screen as a percentage.
  - `item_font: "Ubuntu Regular 20"`: The font used for menu items.
  - `item_color: "#cccccc"`: The color of unselected menu items.
  - `selected_item_color: "#ffffff"`: The color of the currently selected menu item.
  - `icon_width, icon_height`: The dimensions for any icons associated with menu items.
  - `item_icon_space`: The spacing between the icon and the text for each item.
  - `item_height, item_padding, item_spacing`: The height of individual items, internal padding, and spacing between items.
  - `selected_item_pixmap_style: "select_*.png"`: An image or style used to visually indicate the selected item.

3. Countdown Label

- `+ label { ... }` This defines the countdown text that appears before the operating system starts booting automatically:
    - `left, top`: The position of the text on the screen.
    - `align: "center"`: Centers the text.
    - `id: "__timeout__"`: A special identifier for the countdown.
    - `text: "Selected OS will start in %d seconds"`: The text displayed for the countdown. `%d` will be replaced by the remaining seconds.
    - `color: "#cccccc"`: The color of the text.
    - `font: "Ubuntu Regular 17"`: The font type and size used for the countdown text.

 Note:

All these settings provide complete control over the appearance of the GRUB menu, from backgrounds and fonts to how selected items and the countdown are displayed. Any value can be modified to change the boot screen's look and feel to align with your distribution's identity.

2.7.4.2 Exploring the `/usr/share/backgrounds` Directory

After customizing the GRUB interface visually, it's time to dive into a core element of the graphical user experience: system backgrounds.

The `/usr/share/backgrounds` directory holds the images and wallpapers that can be displayed in the desktop environment. These can include default backgrounds shown at login or images that users can select later.

Understanding the contents of this directory and how it's organized empowers you to:

- **Visually Customize the Distribution's Identity:** Select backgrounds that reflect the character of Helwan Linux, giving users a distinct feel from the outset.
- **Manage Performance:** Using appropriately sized and compressed images reduces memory consumption and speeds up desktop loading.
- **Provide User Options:** Enable users to easily change wallpapers through desktop settings.

2.7.4.3 Exploring the `/usr/local` Directory

Having explored crucial directories like `/usr/share`, which houses general system files, wallpapers, and GRUB themes, we now turn our attention to `/usr/local`. This directory is vital for any custom distribution.

`/usr/local` is designated for installing programs and tools added by the developer or the user themselves, separate from the official packages managed by the package manager (Pacman). Utilizing this directory empowers you to:

- **Customize the Distribution with Additional Tools:** Install specific programs or scripts for your distribution without affecting the main package system.
- **Isolate Local Software from the Core System:** This minimizes the risk of package conflicts and simplifies update management.
- **Provide a Flexible Environment for Users and Developers:** Users can add their own tools, and developers can test applications without jeopardizing the core system libraries.

In the upcoming sections, we will detail how to organize this directory, the most important files it should contain, and how to add your distribution's specific tools in an organized and secure manner.

2.7.4.3.1 Exploring the `/usr/local/bin` Directory

The `/usr/local/bin` directory is the beating heart of user-specific or developer tools and programs within your distribution. While official packages contain system programs in `/usr/bin`, `/usr/local/bin` is dedicated to software and scripts that you add yourself or that come with the distribution in a customized way, separate from the main package manager (Pacman).

Significance of This Directory:

- **Installing Distribution-Specific Tools:** Every script or program designed specifically for the Helwan Linux distribution can be placed here to ensure easy access and use.
- **Facilitating Maintenance and Updates:** Separating local tools from system programs reduces conflicts and simplifies the process of updating the system or software.
- **Providing a Flexible Development Environment:** It allows developers and users to experiment with their scripts or tools without risking the core system libraries.

In this section, we will explain how to organize this directory, how to set permissions for tools and scripts, and provide practical examples of your distribution's tools that can be placed here to be directly available to all users.

Exploring the `/usr/local/bin/choose-mirror` Script

This script, though simple, is crucial for your distribution. It's responsible for automatically updating the mirror list for Arch Linux repositories during boot, ensuring faster and more stable package downloads based on the user's location.

1. Purpose of the Script

- It **determines which mirror** will be used by the user or the distribution for downloading packages.
- It can be configured to **automatically select mirrors** based on location or command-line parameters.
- It **preserves an original copy** of the mirror list (as `mirrorlist.orig`) to prevent data loss.

2. Step-by-Step Code Explanation

Bash
```
get_cmdline() {
    local param
    for param in $(</proc/cmdline); do
        case "${param}" in
            "${1}="*)
                echo "${param##*=}"
                return 0
                ;;
        esac
    done
    return 1 # Indicate failure if parameter not found
}
```

- `get_cmdline()`: This is a **function** responsible for reading the kernel command line at boot, located in `/proc/cmdline`.
- **Functionality:** It searches for a specific value associated with a given parameter, such as `mirror=`.
- **Explanation:**
    - `for param in $(</proc/cmdline)`: Iterates through each word found on the `/proc/cmdline`.
    - `case "${param}" in "${1}="*)`: Checks if the current parameter starts with the desired argument (passed as `$1`).
    - `echo "${param##*=}"`: If it matches, this extracts and outputs the value *after* the equals sign (`=`).
    - `return 0`: Exits the function successfully after finding the value. (Added `return 1` for clarity when the parameter isn't found).

Bash
```
mirror="$(get_cmdline mirror)"
[[ "$mirror" == 'auto' ]] && mirror="$(get_cmdline archiso_http_srv)"
```

```
[[ -n "$mirror" ]] || exit 0
```

- **First line:** Attempts to read the `mirror` value directly from the command line.
- **Second line:** If the `mirror` value is set to `auto`, it then tries to obtain the server address from the `archiso_http_srv` parameter.
- **Third line:** If no mirror has been determined (i.e., `$mirror` is empty), the script **exits without making any changes** to the `mirrorlist`.

Bash
```
mv /etc/pacman.d/mirrorlist /etc/pacman.d/mirrorlist.orig
```

This command creates a **backup** of the current mirror list, naming it `mirrorlist.orig`. This is essential to prevent the loss of original user configurations and allows for easy restoration if needed.

Bash
```
cat >/etc/pacman.d/mirrorlist <<EOF
#
# Arch Linux repository mirrorlist
# Generated by archiso
#

Server = ${mirror%%/}/\$repo/os/\$arch
EOF
```

This section **writes a new mirror list** to `/etc/pacman.d/mirrorlist`.

- `${mirror%%/}`: This shell parameter expansion **removes any trailing forward slashes (`/`)** from the mirror URL to prevent potential errors.
- `\$repo` and `\$arch`: These are standard `pacman` variables that automatically resolve to the repository name (e.g., `core`, `extra`) and the system's architecture (e.g., `x86_64`), respectively.

Result: Your system is now configured with a specific server to use for downloading packages.

3. Importance of This Script in the Distribution

- **Accelerated Package Downloads:** Selecting the closest and fastest mirror significantly reduces the time required to download packages. ⚡
- **Flexibility:** The mirror can be easily specified from the command line during boot for customized setups.
- **Safety:** Saving an original copy of the mirror list protects the system from losing its configured mirrors.
- **User Experience:** It streamlines the initial setup process for the distribution, particularly in Live CD or USB environments.

4. Practical Example

If you want to specify a mirror during boot:

```
mirror=http://mirror.example.com
```

Or to have the system choose the server automatically:

```
mirror=auto
```

The script will automatically update `/etc/pacman.d/mirrorlist` before any package installation or update operation.

Exploring the `launch` Script

The `launch` script is responsible for properly initiating the graphical installer, **Calamares**, within the distribution's live environment, particularly when utilizing the **Copy-to-RAM** feature. This script ensures that the system, root, and kernel paths are correctly configured before the installation process begins.

1. Defining Core Variables

- `DIR="/etc/calamares"`
- `KERNEL=\uname -r``
    - `DIR`: Points to the location of Calamares' configuration files within the system.
    - `KERNEL`: Stores the current kernel version using the `uname -r` command. This is crucial for specifying the correct kernel path during installation.

2. Checking for Copy-to-RAM Presence

Bash
```
if [[ -d "/run/archiso/copytoram" ]]; then
```

The script checks if the system was booted using the `copytoram` option, which involves copying the entire distribution into RAM. If this directory exists, it signifies that the original files on the media (USB/CD) will not be used directly, and the paths within Calamares must be adjusted.

3. Modifying File Paths in `unpackfs.conf`

Bash
```
sudo sed -i -e
's|/run/archiso/bootmnt/arch/x86_64/airootfs.sfs|/run/archiso/copytoram/airoo
tfs.sfs|g' "$DIR"/modules/unpackfs.conf
```

This command changes the path to `airootfs.sfs` to the version residing in RAM. `airootfs.sfs` is the system's ready-to-use root filesystem image that the installer requires to deploy the distribution.

Bash
```
sudo sed -i -e "s|/run/archiso/bootmnt/arch/boot/x86_64/vmlinuz-linux|/usr/lib/modules/$KERNEL/vmlinuz|g" "$DIR"/modules/unpackfs.conf
```

This line updates the path to the system's kernel (`vmlinuz-linux`) within the configuration file, ensuring that Calamares uses the correct kernel located within the system rather than the one on the boot media.

4. Launching Calamares

Bash
```
sudo -E calamares -d
```

- `sudo -E`: Preserves existing environment variables when running the script with root privileges.
- `-d`: Launches Calamares in debug mode, displaying all diagnostic messages.

Result: Calamares starts correctly and is ready to install the distribution onto the target machine.

5. Script Significance

- **Copy-to-RAM Compatibility:** Ensures Calamares functions correctly when the entire system is copied to RAM.
- **Path Error Prevention:** Avoids any errors that might arise from incorrect paths for the root filesystem or the system kernel.
- **Ease of Maintenance and Development:** Separating custom tools makes it simpler to modify paths or update the script later.
- **Smooth User Experience:** Guarantees that the graphical installer launches without complications, even in Live USB/CD environments.

Exploring the `/usr/local/bin/livecdsound` Script

The `livecdsound` script is a critical utility for managing audio within a Live CD or Live USB environment of **Helwan Linux**. It ensures that audio works immediately after boot, providing a seamless and ready-to-use experience for users. The script handles unmuting, volume adjustment, and selection of the default sound card, supporting a wide range of hardware.

---

1. Main Purpose of the Script

- **Unmute all connected sound cards** to ensure sound is audible without manual adjustments.
- **Set volume levels** for essential channels (Front, Master, Headphone, PCM, DAC, Synth, etc.) according to typical defaults.
- **Allow selection of the primary sound card** if multiple are available.
- **Provide audible cues** to guide the user in selecting the default audio device.
- **Ensure compatibility with different hardware**, including Intel HDA, SB Live, VIA, and other sound cards.

---

2. Key Components and Functions

a. Helper Functions

1. **`usage()`**
   Displays how to use the script and the available options:
   - `-u, --unmute`: Unmute all sound cards
   - `-p, --pick`: Select a card for audio playback
   - `-h, --help`: Display the help message
2. **`bugout()`**
   Prints an error message if a function is called incorrectly or if required data is missing.
3. **`echo_card_indices()`**
   Reads the index numbers of all available sound cards from `/proc/asound/cards`.

---

b. Sound Control per Card

1. **`unmute_and_set_level(card, control, level)`**
   - Unmutes a given channel on a specific card and sets the volume level.
   - Example: `unmute_and_set_level 0 Master 80%`
2. **`mute_and_zero_level(card, control)`**
   - Mutes a channel and sets its volume to 0%.
3. **`switch_control(card, control, state)`**
   - Switches a control on or off (e.g., "Master Playback Switch").
4. **`sanify_levels_on_card(card)`**
   - Sets default levels for multiple channels to ensure audio works consistently.
   - Covers common hardware variations: Intel HDA, SB Live, VIA, etc.
   - Unmutes important channels like Front, Master, PCM, Headphone, DAC.
   - Mutes or disables unnecessary channels like Mic, IEC958 Capture Monitor.
5. **`sanify_levels(card|all)`**
   - Applies `sanify_levels_on_card()` to a single card or all cards.
6. **`list_non_pcsp_cards()`**
   - Returns the list of PCM-capable cards, excluding internal PC speakers.

7. **`unmute_all_cards()`**
    - Unmutes all available sound cards by calling `sanify_levels all`.

---

c. Default Card Management

1. **`is_numeric()`**
    - Checks if a given input is a numeric card index.
2. **`set_default_card(card)`**
    - Updates `/etc/asound.conf` with the selected default card using a template file.
3. **`play_on_card(card, file)`**
    - Plays a sound file on the specified card for feedback or notifications.
4. **`pick_a_card()`**
    - If multiple usable sound cards exist, plays sounds to guide the user in selecting the default card.
    - Waits for input to set the chosen card as default.
    - Provides audible cues like “beep” and “pick-a-card” sounds.

---

3. Runtime Options

The script can be invoked with the following options:

Option	Function
`-h` / `--help`	Show the help message
`-u` / `--unmute`	Unmute all sound cards immediately
`-p` / `--pick`	Select the primary sound card for playback

- If no option is passed or an unsupported option is given, the script prints an error and exits.

---

4. Practical Usage Examples

1. **Unmute all cards:**

```
livecdsound -u
```

2. **Select a card for playback:**

```
livecdsound -p
```

3. **Display help message:**

```
livecdsound -h
```

---

5. Importance in the Distribution

- **Immediate Audio:** Ensures users have working sound on Live CD/USB without manual configuration.
- **User Flexibility:** Allows users to select a primary card and customize volume levels.
- **Hardware Compatibility:** Supports various cards (Intel HDA, SB Live, VIA, etc.).
- **Ease of Maintenance:** Placing the script in `/usr/local/bin` makes updates simple and avoids interference with core system binaries.

 Note: This script can be integrated with the Live environment's startup sequence to provide a **fully functional audio experience right at boot**, enhancing user experience from the first second.

Conclusion: Key Contents of the /usr Directory

After exploring the major subdirectories within `/usr`, including `/usr/share/grub/themes/helwan`, `/usr/share/backgrounds`, and `/usr/local/bin`, it is clear that this directory forms the **backbone of your distribution's user environment**.

Key takeaways:

- **Customization & Identity:** Files in `/usr/share` such as GRUB themes and desktop backgrounds define the visual identity of Helwan Linux and shape the first impression for users.
- **User & Developer Tools:** `/usr/local/bin` hosts scripts and utilities like `choose-mirror`, `launch`, and `livecdsound` that enhance functionality, provide flexibility, and maintain separation from core system files.
- **System Efficiency:** Organizing `/usr` properly ensures smooth operation of the desktop environment, software applications, and system tools without touching critical root files.

- **Extensibility:** By understanding and managing `/usr`, you gain full control to add, modify, or remove components, tailoring the system to the needs of your users.

In essence, mastering the contents and structure of `/usr` is **crucial for building a robust, visually consistent, and user-friendly distribution**, making it the central hub for applications, libraries, and essential tools.

Thank you for reaching here!

Hello, fellow developer! Thank you so much for reading this book. If you're reading this page, it means you've reached the end of the current part of our journey.What you have in your hands is only the be ginning. This book is a work in progress, and we plan to add more chapters to make it a complete reference.

Here's a glimpse of what we're working on now:

The Practical Side: Step-by-step instructions for building the distro.

Calamares: A complete guide to integrating the system installer.

Appendices: Quick guides to arch commands, the hpm tool, and technical terminology.

I believe that perfection is the enemy of progress, and that's why I decided to share this content with you now, so we can learn and build together.

If you have any feedback or ideas, please don't hesitate to share them with us.

We consider you a partner in this project. You can contribute via GitHub [https://github.com/helwan-linux/archiso-book] or on the distro's site at [https://helwan-linux.github.io/helwanlinux/index.html].

I look forward to seeing what you'll build!