

# **OPEN SOURCE ENGINEERING**

**Name:** M.Nandhan

**Student ID:** 2400040295

---

# 1 Understanding the Core Ubuntu Linux Distribution

## 1.0.1 1. Overview and Philosophy

Ubuntu is a powerful, free, and open-source operating system built upon the stable foundation of Debian Linux. It stands as the world's most popular Linux distribution for desktop use, successfully blending cutting-edge features with unparalleled user-friendliness. Developed and maintained by Canonical Ltd., Ubuntu's guiding principle is "Linux for human beings." This philosophy drives its commitment to accessibility, stability, and providing an intuitive computing experience for everyone, from novice users to seasoned developers.

## 1.0.2 2. The Desktop Experience (GNOME)

The standard Ubuntu desktop utilizes the **GNOME** desktop environment, which presents a modern, clean, and highly efficient graphical interface. The key design elements include a permanent dock (launcher) on the left side for quick access to essential applications, and the **Activities Overview**. This view, easily accessed by pressing the Super (Windows) key, provides a centralized hub for managing all open windows, workspaces, and system-wide searching. This streamlined workflow makes Ubuntu feel contemporary and ensures high productivity. Furthermore, Ubuntu is recognized for its strong, out-of-the-box hardware detection and compatibility, simplifying the setup process for most users.

## 1.0.3 3. Software Management and Packaging

Ubuntu employs a robust dual-system for software management. The traditional and reliable **Advanced Packaging Tool (APT)** manages **DEB** packages, handling core system utilities and standard applications sourced from official repositories. Complementing this is the use of **Snaps**, a modern, containerized package format pioneered by Canonical. Snaps bundle an application with all its required dependencies, guaranteeing consistent performance across different Ubuntu versions. Crucially, Snaps run in a **sandboxed** environment, isolating them from the rest of the operating system to significantly enhance overall application security. This flexibility ensures users have access to a vast, up-to-date, and secure software library.

---

## 2 Encryption and GPG

### 2.1 Types of Encryption in Ubuntu

Ubuntu offers two primary approaches to encryption: **Full Disk Encryption (FDE)** and **File/Directory Encryption**.

#### 2.1.1 1. Full Disk Encryption (FDE)

- **What it is:** FDE encrypts the entire hard drive or a large partition, including the operating system files, swap space, and user directories.
- **How it works:** Ubuntu uses **LUKS** (Linux Unified Key Setup) for FDE. When the system boots, you are prompted for a **passphrase**. If correct, LUKS decrypts the entire drive, and the decryption process runs transparently in the background while the system is in use.
- **Purpose:** The primary defense against data loss due to **theft** or **physical access** to the computer when it is turned off. If someone steals the hard drive, the data is useless without the LUKS passphrase.
- **Implementation:** FDE is typically enabled during the Ubuntu installation process by selecting the "Encrypt the new Ubuntu installation" option. It's much more difficult to enable after installation.

#### 2.1.2 2. File and Directory Encryption

- **What it is:** This method encrypts specific files, directories, or messages, offering granular control over which data is protected.
- **Tools:**
  - **GPG (GNU Privacy Guard):** The standard, used for encrypting individual files and especially for secure communication using **public-key cryptography**.
  - **eCryptfs (older):** Previously used for encrypting the user's Home directory, but has been largely phased out for FDE.

### 2.2 GPG (GNU Privacy Guard) Explained

**GPG** is the GNU implementation of the **OpenPGP** standard (originally Pretty Good Privacy - PGP). It is essential for protecting individual files and ensuring secure, authenticated communication.

---

### 2.2.1 1. Core GPG Concepts

GPG relies on **asymmetric cryptography**, which uses a pair of mathematically linked keys:

- **Public Key:** This key is shared with everyone. It can be used to **encrypt** a message that only you can read, or to **verify** a signature you created.
- **Private (Secret) Key:** This key is kept **secret** and is protected by a strong passphrase. It is used to **decrypt** messages sent to you, or to **digitally sign** files to prove they came from you.

### 2.2.2 2. Basic GPG Command-Line Usage

GPG is usually pre-installed on Ubuntu and is primarily used through the command line (Terminal).

#### A. Generating a Key Pair

The first step is to create your public and private key pair:

Bash

```
gpg --full-generate-key
```

You will be prompted to select the key type (RSA and RSA is common), keysize (4096 is recommended), expiration date, and your Real Name, Email, and a strong **passphrase** to protect your private key.

#### B. Encrypting a File for Yourself (Symmetric Encryption)

To quickly encrypt a file using a single passphrase (like a standard password), use symmetric encryption:

Bash

```
gpg -c myfile.txt
```

This command will prompt you for a passphrase and create an encrypted file named `myfile.txt.gpg`.

#### C. Encrypting a File for Someone Else (Asymmetric Encryption)

To securely send a file, you must use the recipient's **Public Key** (which you must have previously imported into your keyring with `gpg --import`):

Bash

```
gpg --encrypt --recipient "recipient@example.com" mysecretfile.doc
```

This creates `mysecretfile.doc.gpg`. Only the recipient, who holds the corresponding Private Key, can decrypt it.

---

**D. Decrypting a File** To decrypt a file that was encrypted for you:

Bash

```
gpg --decrypt mysecretfile.doc.gpg
```

You will be prompted for the passphrase that protects your Private Key. You can use the `--output` option to specify the decrypted

## 3 Sending Encrypted Email

### 3.1 Prerequisite: Setting Up GPG

Before you can send or receive encrypted mail, both you and your recipient must have GPG keys set up and exchanged:

1. **Generate Keys:** Both parties must have generated a public/private key pair using GPG (as discussed previously, using `gpg --full-generate-key`).
2. **Exchange Public Keys:** You need the recipient's **Public Key**, and they need your Public Key. You can exchange these by:
  - **Exporting** the key: `gpg --armor --export 'Recipient Name' > recipient_key.asc` and sending the `.asc` file.
  - **Uploading** the key to a public key server.
3. **Import Key:** You must import the recipient's key into your GPG keyring: `gpg --import recipient_key.asc`.

### 3.2 Sending the Encrypted Email

The most common and user-friendly way to send GPG-encrypted emails on Ubuntu is by using **Mozilla Thunderbird** with the **Enigmail** add-on (or its built-in equivalent in modern versions of Thunderbird).

#### 3.2.1 1. Compose the Message

- **Open Thunderbird** and start composing a new email.
- Write your message as usual.

---

### 3.2.2 2. Encryption and Signing

You will use the GPG function built into the mail client to perform two critical steps:

1. **Encryption:** You must encrypt the email using the **recipient's Public Key**. Only their corresponding **Private Key** can decrypt it. If you have multiple recipients, you must encrypt the message using the Public Key of *every single recipient*.
2. **Digital Signature:** You **sign** the email using **your Private Key**. This allows the recipient to verify that the email truly came from you and has not been tampered with in transit.

In Thunderbird, this is typically done by clicking a dedicated **OpenPGP or Security** menu or button within the compose window and ensuring both the "**Encrypt**" and "**Sign**" options are checked.

### 3.2.3 3. Verification and Sending

- The client will check that you have the required **Public Key** for the recipient(s). If a key is missing, it will warn you.
- When you click **Send**, Thunderbird uses GPG to encrypt the message body and attach your digital signature before transmitting the scrambled data.

### 3.2.4 4. Recipient's Experience (Decryption)

1. The recipient receives the scrambled email.
2. Their email client automatically uses their **Private Key** (protected by their passphrase) to decrypt the message contents, revealing the original text.
3. Their client simultaneously uses your **Public Key** to verify the digital signature, confirming the email's authenticity.

## 4 Privacy Tools From Prism Break

### 4.0.1 1. Tor Browser (Web Browsers / Anonymizing Networks)

- **What it is:** A web browser built on Firefox that routes your internet traffic through the Tor network, a volunteer-operated network of relays.
- **Privacy Focus:** Provides **strong anonymity** by obscuring your IP address and location from the websites you visit. It also includes anti-fingerprinting measures.

- 
- **PRISM Break Note:** PRISM Break strongly recommends using Tor Browser for all web surfing when maximum anonymity is required.

#### 4.0.2 2. Debian (Operating Systems)

- **What it is:** A popular and highly ethical GNU/Linux distribution known for its strict adherence to Free Software principles and ethical manifesto.
- **Privacy Focus:** Unlike proprietary operating systems like Windows and macOS (which PRISM Break generally avoids), Debian is fully open-source, allowing for audits. It has a long tradition of software freedom and transparency.
- **PRISM Break Note:** It's recommended as a top GNU/Linux choice for users transitioning from proprietary systems, highlighting its commitment to free software and its stable nature.

#### 4.0.3 3. Thunderbird (Email Clients)

- **What it is:** A free, open-source, and cross-platform email client developed by Mozilla.
- **Privacy Focus:** Thunderbird is the top choice for desktop email due to its open-source nature and its long-standing **native support for OpenPGP** (GPG) encryption and digital signatures. This allows users to easily encrypt and authenticate their emails end-to-end.
- **PRISM Break Note:** It is highly recommended for securely managing email with built-in PGP features.

#### 4.0.4 4. KeePassXC (Password Managers)

- **What it is:** A free, open-source, and cross-platform password manager.
- **Privacy Focus:** It stores all your passwords in a single, highly encrypted database file that is stored **locally** on your device, giving you total control over your sensitive data. It does not rely on a cloud service.
- **PRISM Break Note:** It is preferred for its strong encryption, open-source license, and local-only storage, minimizing exposure to third-party services.

#### 4.0.5 5. Firefox (Web Browsers)

- **What it is:** A fast, flexible, and secure web browser developed by the non-profit Mozilla Foundation.

- 
- **Privacy Focus:** Firefox is open-source and provides extensive privacy controls, including enhanced tracking protection (ETP), container technology, and a robust add-on ecosystem for further hardening security (like uBlock Origin).
  - **PRISM Break Note:** While Tor Browser is for anonymity, Firefox is the recommended alternative for general web use when a site doesn't work well with Tor, provided the user configures its settings and replaces the default search engine with a privacy-focused one.

## 5 Open Source License

Certainly. Here is the information about the **MIT License** organized into clear, descriptive headings, strictly maintaining a paragraph-only format within each section.

### 5.1 The Core Purpose and Classification

The MIT License is renowned as one of the most permissive and concise open-source licenses currently in use. Originating from the Massachusetts Institute of Technology, its primary goal is to encourage maximum adoption and reuse of software with minimal legal friction. It is formally classified as a **permissive license**, meaning it grants users broad rights to use, modify, and distribute the software without imposing the reciprocal sharing obligations seen in copyleft licenses, such as the GNU General Public License (GPL). This makes the MIT License highly favorable for both commercial enterprises and proprietary software development.

### 5.2 Granted Rights and Permissions

The license grants blanket permission to any individual or entity obtaining a copy of the software and its associated documentation to deal with the Software without restriction. Specifically, users are granted explicit rights to **use, copy, modify, merge, publish, distribute, sublicense, and/or sell** copies of the software. This expansive grant allows developers to incorporate MIT-licensed code into projects that may ultimately be closed-source and sold commercially, provided they meet the few mandated conditions.

### 5.3 The Only Two Conditions for Distribution

Unlike licenses that enforce reciprocal sharing, the MIT License has only two critical requirements that must be met when the software is distributed or included in a larger work. The first condition is the mandatory inclusion of the original **Copyright Notice** (e.g., Copyright <YEAR> <COPYRIGHT HOLDER>).

---

The second is the mandatory inclusion of the full **License Text** itself. If these two simple requirements are satisfied, the user can otherwise treat the code as they wish, including releasing their modifications under a proprietary license.

#### 5.4 Disclaimer of Warranty and Liability

A key component of the MIT License is its comprehensive liability disclaimer, which serves to protect the original authors. The license emphatically states that the software is provided "**AS IS**," meaning it comes without any guarantee or warranty of any kind, whether express or implied, including warranties of merchantability or fitness for a particular purpose. Furthermore, the license explicitly protects the authors and copyright holders, asserting they **shall not be held liable** for any claim, damages, or other liability arising from the use or other dealings in the software. This places the entire risk associated with the software onto the end-user.

## 6 Self Hosted Server

### 6.1 About

4ga Boards is an **open-source, self-hosted project management tool** designed to be a straightforward and intuitive **Kanban board system** for real-time task tracking. Developed to offer a full-featured, less complex alternative to existing proprietary tools, it is licensed under the highly permissive **MIT License**, emphasizing transparency and user control over data.

#### 6.1.1 Key Features

- **Kanban Methodology:** It is based on the visual Kanban board technique, displaying tasks as **cards** across **lists** that represent different stages of a workflow.
- **Structured Workflow:** The system supports a deep organizational hierarchy: **projects - boards - lists - cards - tasks**.
- **Intuitive Design:** Features an elegant **Dark Mode**, a **Simplistic Wide Screen Design** to reduce clutter, and **Collapsible Lists and Sidebar** to maximize screen space for complex projects.
- **Power User Tools:** It includes an **Advanced Markdown Editor**, **Powerful Shortcuts**, and **Multitasking Capabilities** that allow users to simultaneously edit card descriptions while filtering or rearranging the board.

- 
- **Secure Access:** Supports Single Sign-On (**SSO**) via Google, GitHub, and Microsoft for simplified and secure registration and login.
  - **Realtime Updates:** The web application is designed for real-time collaboration, meaning changes are reflected instantly without requiring a page reload.

## 6.2 Installation Process (Docker Compose)

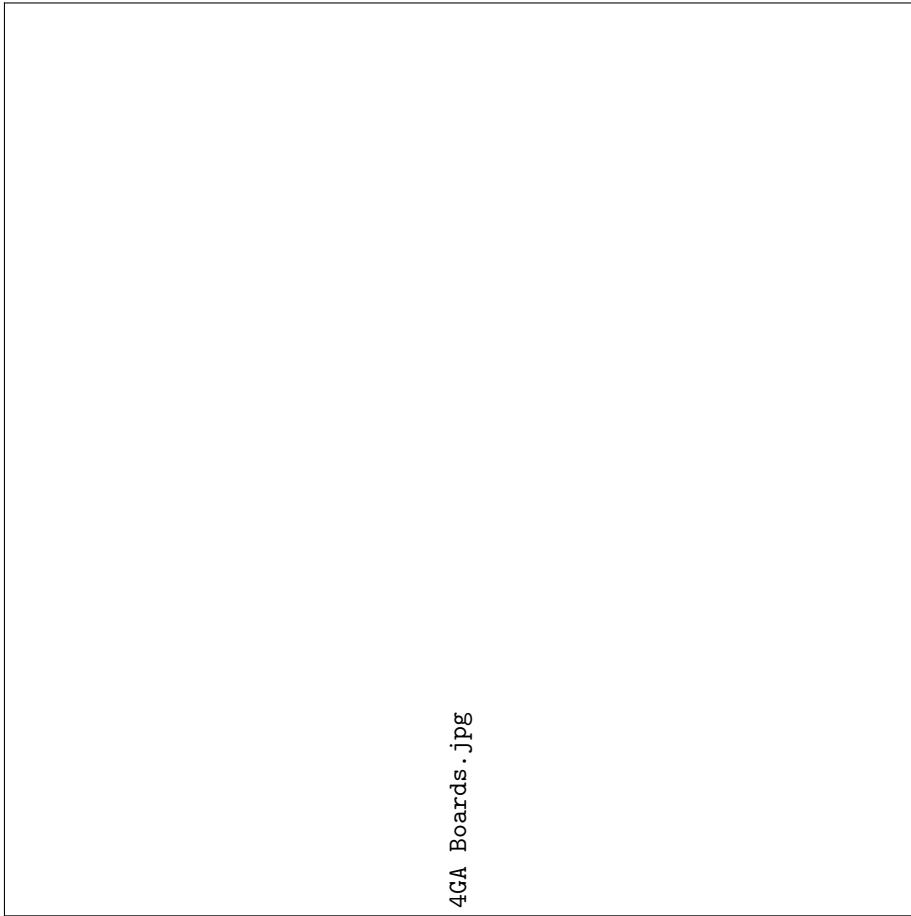
The recommended and easiest method for self-hosting **4ga Boards** is by utilizing **Docker Compose**, a tool that efficiently manages both the application and its dedicated **PostgreSQL** database as integrated, multi-container services. Before starting, two key prerequisites must be met: **Docker** needs to be installed on your hosting machine, and **Docker Compose** must also be available (it is often bundled with Docker Desktop or installed separately).

The installation process begins by **downloading the necessary configuration file**. You use the `curl` command to fetch the official `docker-compose.yml` file and save it in your chosen deployment directory: `.`. Following the download, the configuration file requires a critical security and network update. You must **configure the instance variables** by editing the **environment** sections within the `docker-compose.yml` file: set the `BASE_URL` to your specific domain or public IP (e.g., `http://myboards.com`); define a strong, randomly generated value for the `SECRET_KEY` (which can be generated using `openssl rand -hex 64`); and replace the placeholder with a strong password for the database, setting both `POSTGRES_PASSWORD` and ensuring it matches the password specified in the `DATABASE_URL` environment variable.

Once configured, the next step is to **pull the images and start the services**. By executing the command `docker compose up -d`, Docker Compose downloads the required application and database images and initiates both the **4ga Boards** and PostgreSQL containers in detached mode (`-d`). Finally, **accessing 4ga Boards** is done via the URL defined in your `BASE_URL`. If deploying locally, the default access point is `http://localhost:3000`. The default login credentials for initial access are User: `demo` and Password: `demo`.

## 4GA Boards Resources

Translated Document



4GA Boards.jpg

## 7 Open Source Contribution

### 7.1 PR 1 : First Contribution

#### 7.1.1 Goal

The project's objective is to simplify the standard open-source contribution workflow, allowing beginners to easily add their name to the project's `Contributors.md` file.

#### 7.1.2 The Contribution Workflow

The tutorial details the standard `fork - clone - edit - pull request` sequence, essential for collaborative coding.

---

### 7.1.3 1. Setup

- **Fork:** Create a copy of the repository in your personal GitHub account.
- **Clone:** Download the forked repository to your local machine using the `git clone` command and the SSH URL.
- **Prerequisites:** Ensure **Git** is installed; alternatives for users uncomfortable with the command line (GUI tools) are provided.

### 7.1.4 2. Making Changes

- **Branch:** Create a new isolated branch for your changes using `git switch -c your-new-branch-name`.
- **Edit:** Add your name to the `Contributors.md` file using a text editor.
- **Commit:** Stage the changes with `git add Contributors.md` and save them locally with `git commit -m "Add your-name to Contributors list"`.

### 7.1.5 3. Submission

- **Push:** Upload your local branch to your GitHub fork using `git push -u origin your-branch-name`.
- **Pull Request (PR):** Go to your GitHub repository and submit a PR via the "Compare & pull request" button for review by the project maintainers.

### 7.1.6 Difficulties and Solutions

The guide anticipates and solves two common beginner issues:

- **Old Git Version:** If the `git switch` command fails, use the older command: `git checkout -b your-new-branch`.
- **Authentication Error:** If `git push` fails due to GitHub removing password support, the solution is to configure an **SSH key** or a **Personal Access Token** and ensure your remote URL is set to the **SSH protocol** (`git remote set-url origin git@github.com:....`).

### 7.1.7 Next Steps

Upon merging the PR, the user is encouraged to celebrate their first contribution and seek out other beginner-friendly issues on the project list.

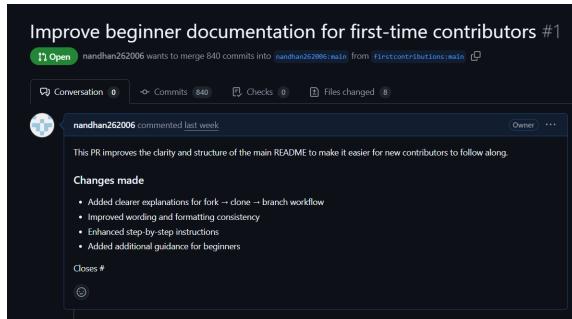


Figure 1: Enter Caption

## 7.2 PR 2 : Puter

Puter is an advanced, free, and open-source platform, effectively functioning as an "Internet OS" that is both exceptionally fast and highly extensible. It is designed to serve multiple critical roles: primarily as a privacy-first **personal cloud** for securely keeping all user files, apps, and games accessible from anywhere; as a robust platform for building and publishing websites and web applications; and as a versatile **remote desktop environment** for managing servers and workstations. It is positioned as a powerful, feature-rich alternative to services like Dropbox and Google Drive, built on principles of open-source transparency.

### 7.2.1 Licensing and Self-Hosting Options

The entire Puter project is distributed under the **AGPL-3.0 license**. This license is highly protective, requiring that any individual or entity who runs the modified software over a network must make the corresponding source code publicly available. For deployment, the project strongly emphasizes self-hosting, providing detailed instructions for several methods: **Local Development** using Node.js (version 20.19.5+ is required) for immediate testing, and easy, reliable deployment using **Docker** or **Docker Compose** on Linux, macOS, and Windows systems. Users require a minimum of 2GB of RAM to run the system smoothly. For immediate access without self-hosting, a live hosted version is also available at Puter.com.

### 7.2.2 Community and Support

Puter maintains a strong connection with its community through various channels for support and contribution. Users can engage directly with maintainers via Discord, Reddit, and X (Twitter). For technical issues, the community is encouraged to report bugs or request features by opening an issue on the project's repository. Specific security concerns can be privately addressed by emailing

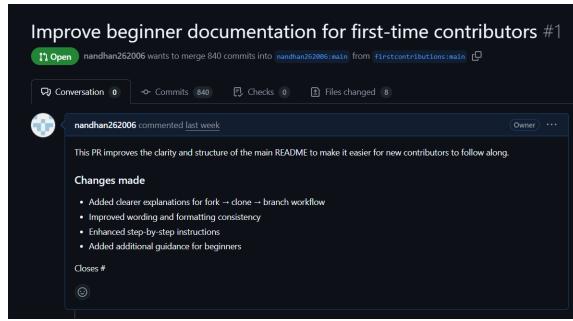
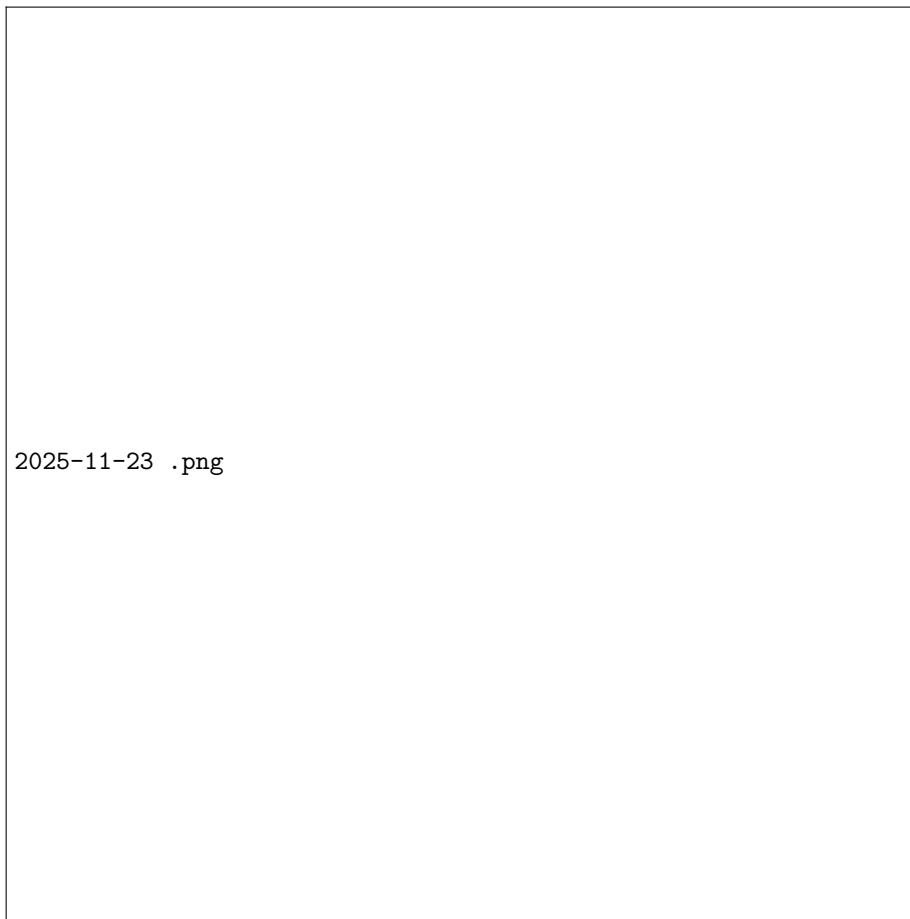


Figure 2: Enter Caption

`security@puter.com`, ensuring a responsive and accountable support structure.



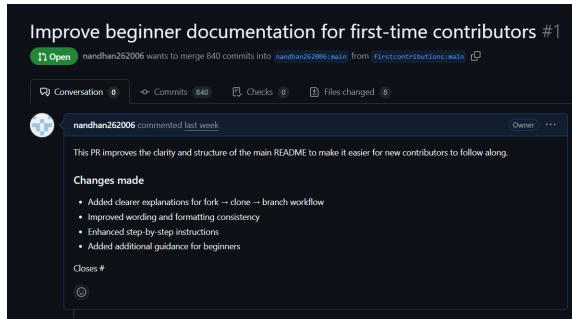


Figure 3: Enter Caption

## 7.3 PR 3 : Y24 Open Source Engineering

### 7.3.1 Introduction and Purpose

4GA Boards are modern **Embedded System Boards** designed to function as cost-effective **Self-Hosting Servers**. Their primary objective is to empower users to maintain their data independently of external cloud services, thereby maximizing **data privacy and security**. This provides an opportunity for individuals to host small-scale web or IoT (Internet of Things) applications with full control over the infrastructure.

### 7.3.2 Technical Components

The core server setup relies on the 4GA Board, which features an **ARM Cortex-based processor**. Essential components include Micro SD/SSD storage, network **connectivity** via Ethernet or Wi-Fi, and a dedicated **power supply**. The entire system is built upon a Linux-based operating system, such as **Ubuntu Server** or **Debian**, which provides the necessary server environment.

### 7.3.3 Operation and Usage

The board operates by running a Linux server OS and configuring standard web server software (like Apache or Nginx). Access is typically granted locally via the network or publicly via a domain name. These systems are ideal for **IoT Data Collection**, managing small **database systems**, hosting personal websites, or creating a private file-sharing and **cloud backup server**, offering users complete management control and valuable practical experience.



## 7.4 PR 4: Openllmtry

Here is a short description of the Pull Request, organized into paragraphs with headings.

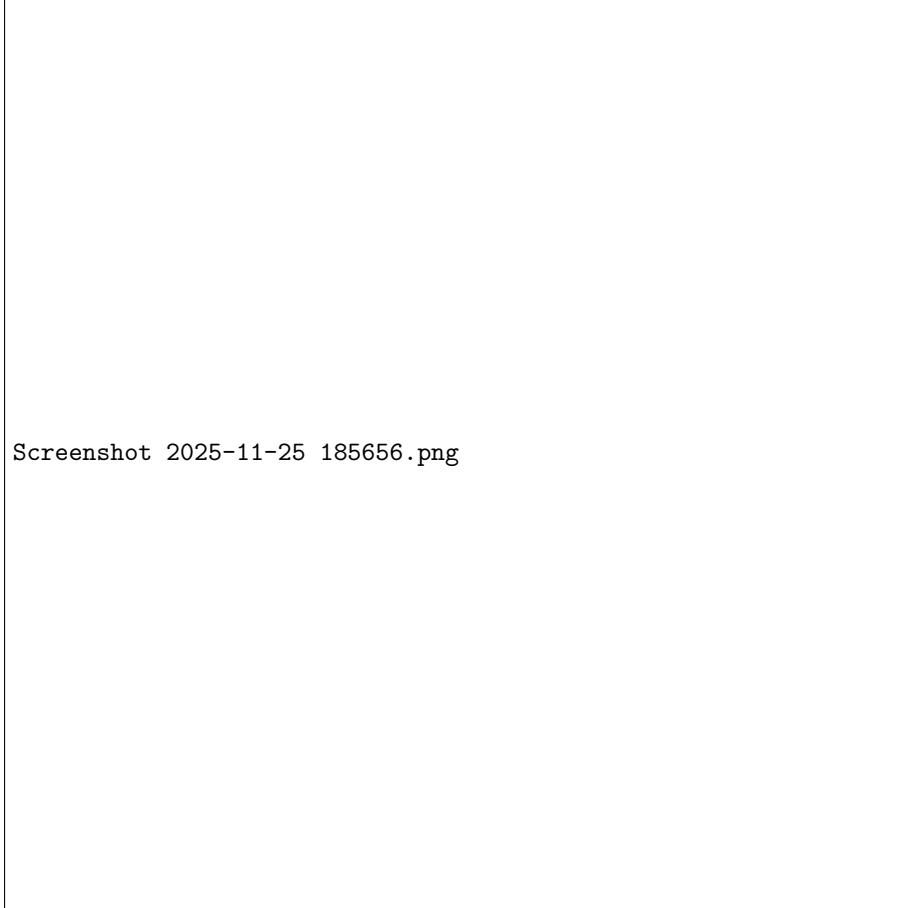
### 7.4.1 The Core Problem

The main difficulty addressed by this Pull Request was the project's rigid dependency management, which lacked selective instrumentation. Users were forced to install a large number of dependencies even if they only needed a small subset (e.g., only openai instrumentation). Furthermore, the activation of the necessary Python virtual environment was inconsistent and poorly supported across different command-line shells, such as PowerShell, Bash, and Fish, creating friction for developers using varied environments.

---

#### **7.4.2 The Solution: Selective Installation and Universal Activation**

This Pull Request resolves these issues through two critical updates. First, it implements Selective Dependencies by updating setup.py to utilize extras require. This change allows users to install specific, necessary groups of instrumentation, such as full, minimal, openai, or langchain, dramatically improving efficiency and reducing clutter. Second, it ensures Universal Virtual Environment Activation by adding dedicated activation scripts (Activate.ps1, activate.csh, activate.fish, etc.) for multiple shell environments. These scripts standardize the process of environment activation, correctly configuring variables like PATH and PYTHONHOME, thereby providing consistent setup across all developer environments.



Screenshot 2025-11-25 185656.png

---

## 7.5 PR 5 : Google Colab IO instructions to pandas IO documentation

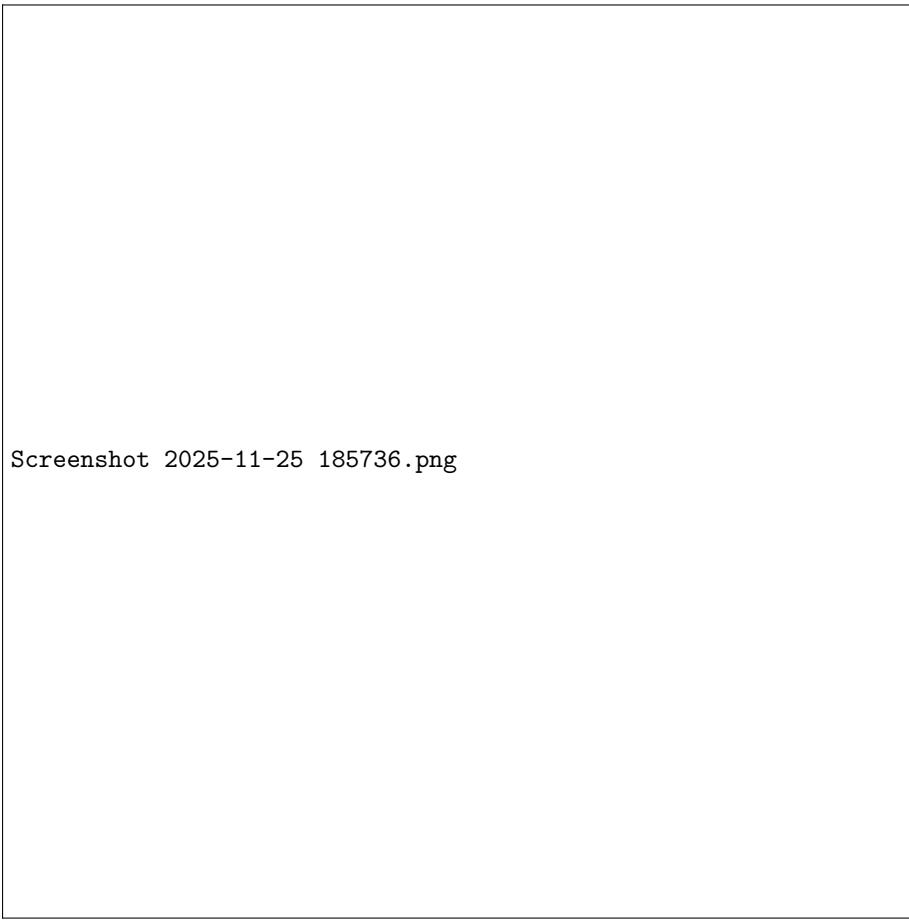
### 7.5.1 The Issue (What was Missing)

The core issue was a significant gap in the **pandas Input/Output (IO) documentation**. Specifically, the documentation did not contain instructions or guidance for users performing common data loading and saving tasks while operating within a **Google Colaboratory (Colab)** environment. This omission created unnecessary difficulty for a large segment of the data science community that uses Colab notebooks extensively for running pandas code.

### 7.5.2 The Solution (What Was Added)

The solution was to directly address this gap by adding dedicated instructions. The change consists of adding **Google Colab IO instructions** to the pandas IO documentation (`doc/source/whatsnew/io.rst`). This update includes clear guidance and specific code examples on how to correctly handle data files (e.g., mounting Google Drive, uploading files) when working with pandas within a Colab environment.

This ensures the documentation is more complete, directly supports the platform used by many pandas users, and improves the overall usability of the pandas library in a cloud notebook setting.



## 7.6 PR 6 : Fix/ Docs Formatting

### 7.6.1 The Issue (The Problem Being Fixed)

The core issue addressed by this Pull Request is a **bug fix** related to the **mis-formatting and incorrect line breaks** present across various documents within the ZenML project (referenced by the title `docs: fix mis-formatting and line breaks across docs (zenml-io#4084)`). This type of formatting error often leads to poor readability, rendering issues on the documentation website, or inconsistent presentation of instructions and explanations for users. The fix falls under a **bug fix (non-breaking change)** category, meaning it corrects existing incorrect behavior without introducing new features or breaking existing functionality.

---

### **7.6.2 The Solution (What Was Done)**

The solution provided by involves changes that correct the mis-formatting within the documentation source files. Although the exact code changes are not detailed, the PR explicitly targets fixing issues related to mis-formatting and line breaks. A key process point during the submission involved ensuring the pull request was correctly targeted at the `develop` branch instead of the temporary `main` branch, which required communication between the author and the contributor (`bcdurak`). After resolving the target branch issue and deleting a duplicate PR, the submission is ready for final review and merge, having passed security and licensing checks.



---

## **8 Linkedin Post Links**

### **8.1 PR :**

[https://www.linkedin.com/posts-b-87993b389Opensource-github-developerjourney-activity-739907800323utm\\_source=share&utm\\_medium=member\\_desktop&rcm=ACoAAF-2fh0Bnm\\_Z20VLqqUsvN790REF\\_QOECsY](https://www.linkedin.com/posts-b-87993b389Opensource-github-developerjourney-activity-739907800323utm_source=share&utm_medium=member_desktop&rcm=ACoAAF-2fh0Bnm_Z20VLqqUsvN790REF_QOECsY)

### **8.2 Journey Of Open Source :**

[https://www.linkedin.com/posts/nandhan-b-87993b389\\_activity-7399075775599091712-5QJ4?utm\\_source=share&utm\\_medium=member\\_desktop&rcm=ACoAAF-2fh0Bnm\\_Z20VLqqUsvN790REF\\_QOECsY](https://www.linkedin.com/posts/nandhan-b-87993b389_activity-7399075775599091712-5QJ4?utm_source=share&utm_medium=member_desktop&rcm=ACoAAF-2fh0Bnm_Z20VLqqUsvN790REF_QOECsY)

### **8.3 Self Hosted Project :**

[https://www.linkedin.com/posts/nandhan-b-87993b389\\_kluniversity-h-te-opensource-activity-73990751939utm\\_source=share&utm\\_medium=member\\_desktop&rcm=ACoAAF-2fh0Bnm\\_Z20VLqqUsvN790REF\\_QOECsY](https://www.linkedin.com/posts/nandhan-b-87993b389_kluniversity-h-te-opensource-activity-73990751939utm_source=share&utm_medium=member_desktop&rcm=ACoAAF-2fh0Bnm_Z20VLqqUsvN790REF_QOECsY)