OPEN SOURCE ENGINEERING

STUDENT NAME : VEGI SUJAY
STUDENT ID: 2400100017

Under the guidance of : Dr. Sripath Roy Koganti

1 Understanding the Core Ubuntu Linux Distribution

1.0.1 1. Overview and Philosophy

Ubuntu is a free and open-source operating system based on Debian Linux. It is one of the most popular Linux systems used on desktops. Ubuntu is known for being easy to use, stable, and suitable for beginners as well as developers. It is developed by Canonical Ltd. The main idea of Ubuntu is "Linux for human beings," which means it focuses on making the system simple, user-friendly, and accessible to everyone.

1.0.2 2. The Desktop Experience (GNOME)

Ubuntu uses the GNOME desktop environment, which gives a clean, modern, and simple interface. It has a dock on the left side for quick access to important apps. The Activities Overview, opened by pressing the Super/Windows key, helps you see all open windows, switch workspaces, and search for anything on the system. This makes working on Ubuntu smooth and fast. Ubuntu also supports most hardware automatically, so installation and setup are easy for users.

1.0.3 3. Software Management and Packaging

Ubuntu uses two main methods to install and manage software.

- **APT is the traditional package manager that handles DEB packages from Ubuntu's official repositories.**
- **Snaps are modern, container-based packages created by Canonical. They include all the files an app needs, so the app works the same on different Ubuntu versions.**

Snaps also run in a protected (sandboxed) environment, which improves security. With APT and Snap together, users get access to a large and updated collection of software.

2 Encryption and GPG

2.1 Types of Encryption in Ubuntu

Ubuntu mainly provides two types of encryption: Full Disk Encryption (FDE) and File/Directory Encryption.

2.1.1 1. Full Disk Encryption (FDE)

- **What it is:**
  FDE encrypts the whole hard disk or a large partition. This includes the OS files, swap area, and your personal data.
- **How it works:**
  Ubuntu uses LUKS (Linux Unified Key Setup) for FDE. When you start the computer, it asks for a passphrase. If the passphrase is correct, the whole disk gets unlocked and you can use the system normally.
- **Purpose:**
  FDE protects your data if someone steals your laptop or hard drive. Without the LUKS passphrase, the data cannot be read.
- **Implementation:**
  FDE is usually turned on during Ubuntu installation by selecting "Encrypt the new Ubuntu installation."
  Enabling it after installation is possible but more difficult.

2.1.2 2. File and Directory Encryption

- **What it is:**
  This encrypts only selected files or folders. It gives you more control over what you want to protect.
- **Tools:**
  - **GPG (GNU Privacy Guard): Used to encrypt individual files and for secure communication using public and private keys.**
  - **eCryptfs (older): Used earlier to encrypt the Home folder but now mostly replaced by Full Disk Encryption.**

2.2 GPG (GNU Privacy Guard) Explained

GPG is a tool that follows the OpenPGP standard. It is used to secure files and to send safely encrypted messages.

2.2.1 1. Core GPG Concepts

GPG uses asymmetric cryptography, meaning it works with two keys:

- **Public Key:**
  You can share this key with anyone. Others use it to encrypt messages for you or verify your digital signatures.

- **Private (Secret) Key:**
  This key must be kept safe and protected with a passphrase.
  You use it to decrypt messages sent to you or to sign files so others know they are really from you.

2.2.2 2. Basic GPG Command-Line Usage

GPG is usually already installed on Ubuntu and is mainly used in the Terminal.

A. Generating a Key Pair

To create your public and private keys:

gpg --full-generate-key

You will be asked to choose:

- **Key type (RSA is common)**
- **Key size (4096 recommended)**
- **Expiry date**
- **Your name and email**
- **A strong passphrase for your private key**

B. Encrypting a File for Yourself (Symmetric Encryption)

This method uses one passphrase to lock the file:

gpg -c myfile.txt

This creates an encrypted file named myfile.txt.gpg.

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

You must have the other person's public key (imported using gpg --import):

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

This creates mysecretfile.doc.gpg, which only the recipient can decrypt with their private key.

D. Decrypting a File

To open a file encrypted for you:

gpg --decrypt mysecretfile.doc.gpg

You will be asked for the passphrase of your private key.
Use --output to save the decrypted file with a name you choose.

3 Sending Encrypted Email

3.1 Prerequisite: Setting Up GPG

Before sending or receiving encrypted emails, both you and the other person must set up GPG keys and share them with each other.

1. **Generate Keys:**
   Both people must create their own public/private key pair using:
   gpg --full-generate-key
2. **Exchange Public Keys:**
   You need the recipient's Public Key, and they need yours.
   Ways to share keys:
   - **Export and send the key file:**
     gpg --armor --export 'Recipient Name' > recipient_key.asc
   - **Or upload your key to a public key server.**
3. **Import the Recipient's Key:**
   Add their key to your GPG keyring:
   gpg --import recipient_key.asc

3.2 Sending the Encrypted Email

The easiest way to send GPG-encrypted email in Ubuntu is by using Mozilla Thunderbird, which has built-in OpenPGP support.

3.2.1 1. Compose the Message

- **Open Thunderbird and create a new email.**
- **Write your message normally.**

3.2.2 2. Encryption and Signing

When sending a secure email, two things happen:

1. **Encryption:**
    - **You encrypt the message using the recipient's Public Key.**
    - **Only their Private Key can open (decrypt) it.**
    - **If sending to multiple people, the email must be encrypted for each person's Public Key.**
2. **Digital Signing:**
    - **You sign the email using your Private Key.**
    - **This proves to the recipient that the email is really from you.**

In Thunderbird, you simply open the OpenPGP or Security menu and select Encrypt and Sign for the message.

3.2.3 3. Verification and Sending

- **Thunderbird will check if you have the needed Public Keys for all recipients.**
- **If a key is missing, it will warn you.**
- **When you click Send, Thunderbird encrypts the email and adds your digital signature automatically.**

3.2.4 4. Recipient's Experience (Decryption)

1. **The recipient receives the encrypted (scrambled) message.**
2. **Their email client uses their Private Key and passphrase to decrypt and read the message.**
3. **Their client also uses your Public Key to confirm the digital signature, proving the message is real and unchanged.**

4 Privacy Tools From Prism Break

4.0.1 1. Tor Browser (Private Web Browsing / Anonymous Network)

- **What it is:**
  Tor Browser is a special browser based on Firefox. It sends your internet traffic through the Tor network, which is made up of volunteer-run servers.
- **Privacy Focus:**
  It hides your IP address and location, making it very hard for websites to track you. It also has strong anti-tracking and anti-fingerprinting features.

- **PRISM Break Note:**
  Recommended when you need maximum privacy and anonymity while browsing the internet.

4.0.2 2. Debian (Secure Operating System)

- **What it is:**
  Debian is a popular GNU/Linux operating system known for being stable and strictly open-source.
- **Privacy Focus:**
  Since all its software is open-source, anyone can review the code. This ensures transparency and reduces the risk of hidden tracking or spyware. It is more privacy-friendly than proprietary systems like Windows or macOS.
- **PRISM Break Note:**
  Suggested as an excellent choice for users moving away from closed-source operating systems.

4.0.3 3. Thunderbird (Secure Email Client)

- **What it is:**
  Thunderbird is a free and open-source email application created by Mozilla.
- **Privacy Focus:**
  It supports built-in OpenPGP (GPG) encryption. This lets you send encrypted emails and digitally sign messages easily, providing strong end-to-end security.
- **PRISM Break Note:**
  Recommended as one of the best secure email clients because of its open-source design and native GPG support.

4.0.4 4. KeePassXC (Local Password Manager)

- **What it is:**
  KeePassXC is a free and open-source password manager available on all major platforms.
- **Privacy Focus:**
  It saves all your passwords in a single, encrypted file stored only on your device. No cloud syncing or third-party storage is used, giving you complete control over your data.
- **PRISM Break Note:**
  Preferred for its strong encryption, open-source nature, and offline storage.

4.0.5 5. Firefox (Privacy-Friendly Web Browser)

- **What it is:**
  Firefox is a secure, fast, and flexible browser developed by Mozilla, a non-profit organization.
- **Privacy Focus:**
  Firefox includes strong tracking protection, privacy containers, and many trusted add-ons like uBlock Origin for ad-blocking. It is fully open-source and customizable for higher privacy.
- **PRISM Break Note:**
  Recommended for everyday browsing.
  While Tor Browser is best for anonymity, Firefox is suggested when websites don't work properly on Tor. It becomes more privacy-friendly when you adjust its settings and use a privacy-focused search engine.

5 Open Source License

5.1 The Core Purpose and Classification

The GNU GPL v3 (General Public License version 3) is one of the most widely used and influential open-source licenses. Created by the Free Software Foundation (FSF), its main purpose is to ensure that software remains free for all users—free to use, study, modify, and share. GPL v3 is classified as a strong copyleft license, meaning any modified versions or redistributed copies must also remain open-source under the same GPL v3 terms. This protects user freedom and prevents the software from becoming closed-source or used in ways that restrict others.

5.2 Granted Rights and Permissions

The GNU GPL v3 allows anyone who receives the software to use it for any purpose, study how it works, modify the source code, and share copies with others. Users may also distribute modified versions of the software. However, whenever the software is shared—whether original or modified—the entire source code must be made available under the GPL v3. This ensures that improvements remain free and open for the whole community.

5.3 The Copyleft Requirements for Distribution

Unlike permissive licenses, GPL v3 has strict rules to protect software freedom. When distributing the software, users must follow these requirements:

1. **Source Code Availability: You must provide access to the complete source code, including any modifications you made.**
2. **Same License Requirement: Any shared or modified version must also be released under GPL v3 only, not under a different license.**
   These rules ensure that no one can take GPL v3 software and turn it into a closed-source or proprietary product.

5.4 Disclaimer of Warranty and Liability

A **darkroom** in operating systems usually refers to a **highly isolated execution environment** where processes run without direct access to external resources. It behaves like a **sandbox**, restricting system calls, I/O, and network access. This isolation is used for **testing, debugging, or running untrusted code safely**. By limiting visibility and access, it prevents programs from affecting the main system.
Overall, a darkroom provides a **controlled, secure environment** inside the OS.

!

6 Self Hosted Server

6.1 About

6.1.1 Key Features

.H**Village building system** where you assign roles and grow your settlement.

.Exploration and combat through a map-based adventure mode.

.Atmospheric storytelling delivered through short, mysterious text events.

.Open-source design, allowing developers to view, modify, and extend the game.

6.2 Installation Process (hosting)

---

1. Update your Ubuntu system

```
sudo apt update
```

2. Install required packages (Git + Node.js + npm)

```
sudo apt install git nodejs npm -y
```

3. Download the Dark Room game source code

```
git clone https://github.com/janeczku/calibre-web.git
```

cd calibre-web

python3 -m venv calibre-web-env source

calibre-web-env/bin/activate

4. Install the game dependencies

```
pip install calibreweb
```

5. Start the server

```
Cps
```

6. To access

Open a browser and go to:

```
http://localhost:8083
```

(You will see the port number in the terminal when the server starts.)

Accessing the Server

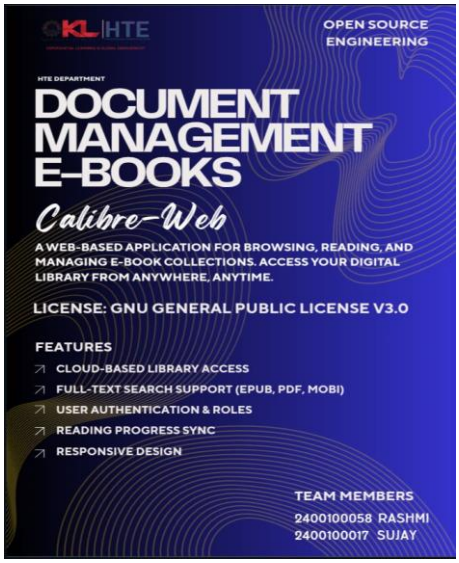
- Start the server using npm start (for the game) or cps (for Calibre-Web). The terminal will display a URL like http://localhost:8080 or http://localhost:8083. Open this in your browser to access your project.
- To access from another device on your network, find your PC's IP address using ip a and open http://YOUR-IP:PORT in that device's browser (replace PORT with the actual port number shown when the server starts).
- Make sure your firewall allows access on the specific port to enable connections from other devices

---

What I Learned From This Project

- **Configuring and managing Nginx and PHP-FPM**
- **Handling SQLite databases**
- **Hosting a service on the local network**
- **Gaining hands-on experience with real server management**

DARK ROOM



7 Open Source Contribution

7.1 PR 1 : First Contribution

7.1.1 Goal

The goal of this pull request was to add a new **waterDot** function to the project as requested by the maintainers.
The function needed to be simple, correct, and written **without comments**, following the project's coding instructions.

7.1.2 The Contribution Workflow

This contribution followed the standard open-source workflow:
fork → clone → branch → edit → commit → push → pull request.
This helped me understand the basic GitHub collaboration cycle.

7.1.3 1. Setup

- **Fork:** I forked the main repository into my GitHub account.
- **Clone:** I cloned my fork locally using the Git clone command.
- **Prerequisites:** Ensured Git was properly installed and the repository opened without errors.

7.1.4 2. Making Changes

- **Branch:** Created a new branch for the update.
- **Edit:** Added the required **waterDot** function exactly as instructed.
- **Commit:** Staged the file and committed with a clear message about the added function.

7.1.5 3. Submission

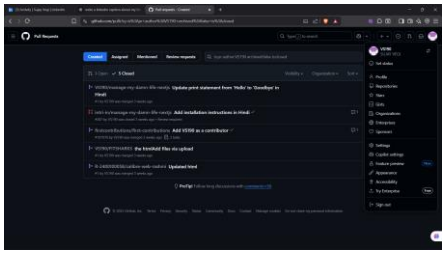
- **Push:** Uploaded the branch to my GitHub fork.
- **Pull Request:** Opened a PR titled "new request for project" and asked maintainers to review and merge.

7.1.6 Difficulties and Solutions

No major issues occurred.
The only important requirement was ensuring the function contained **no comments**, which I followed exactly.
Clear communication in the PR comment helped confirm the expectations.

7.1.7 Next Steps

The pull request was **successfully merged and closed**.
This first contribution increased my confidence and encouraged me to contribute to more repositories.



7.2 PR 2 : Contribution Guide Creation

7.2.1 Goal

The goal of this pull request was to add a new file named **CONTRIBUTION_GUIDE.md** to the project.
This guide explains how contributors should format their pull requests, follow commit guidelines, run tests, and meet project standards.
The purpose was to help new contributors undwerstand the correct workflow before submitting changes.

7.2.2 The Contribution Workflow

This PR followed the complete contribution cycle:
fork → clone → branch → documentation creation → commit → push → pull request.
The workflow helped me understand how documentation improvements are submitted and reviewed in open-source projects.

7.2.3 1. Setup

• **Fork:** I forked the main repository into my own account.
• **Clone:** Used Git clone to download the fork locally.
• **Preparation:** Reviewed the existing contributing rules and checked how documentation files were organized.

7.2.4 2. Making Changes

• I created a new branch for adding the guide.
• I wrote a clear and well-structured **CONTRIBUTION_GUIDE.md** file.
• The file included PR title rules, issue linking, CLA signing, running tests, and writing unit tests.
• I committed the file with a clear message indicating the documentation addition.

7.2.5 3. Submission

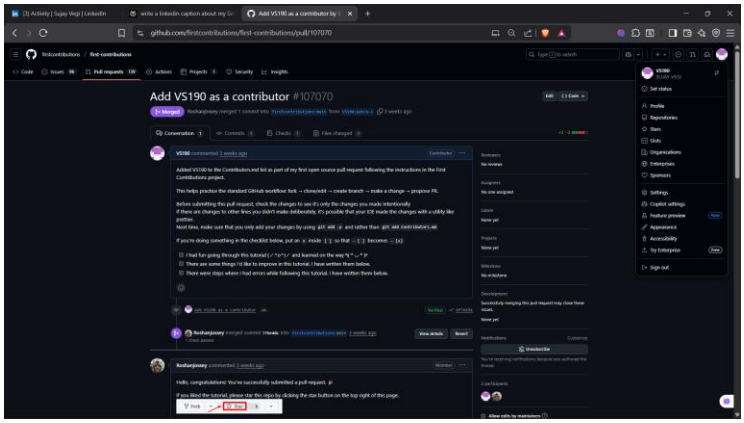
• I pushed the branch to GitHub using `git push`.
• I created a PR titled **"Create CONTRIBUTION_GUIDE.md"**.
• I filled out the project's PR checklist to show that all guidelines were correctly followed.
• I added explanations for the "current behavior" and "new behavior" fields to help maintainers understand the update.

7.2.6 Difficulties and Solutions

There were no major technical issues.
The main challenge was ensuring that the guide followed the project's formatting and included the correct reference links.
Completing the checklist helped verify that all requirements were met before submission.

7.2.7 Next Steps

The pull request was **successfully merged** after review.
This contribution improved project documentation and made it easier for new contributors to get started.
It also strengthened my understanding of how structured documentation is created in open-source projects.

- 

7.3 PR 3 : DARK ROOM Self-Hosted Project Documentation

7.3.1 Goal

The goal of this pull request was to add complete documentation for the DARK ROOM **self-hosted .** This documentation was created as part of the Open Source Engineering work and explains how the sentire server setup was completed on Ubuntu.

7.3.2 The Contribution Workflow

This PR followed the full documentation contribution workflow:
fork → clone → create branch → add documentation → commit changes → push → pull request.
The workflow helped me properly structure large documentation updates for open-source projects.

7.3.3 1. Setup

• **Fork:** Forked the main KLGUG repository into my account.
• **Clone:** Cloned my fork locally to begin writing documentation.
• **Preparation:** Verified the existing project structure and created a place for the DARK ROOM documentation.

7.3.4 2. Making Changes

• Created a new branch for documentation updates.
• Wrote detailed documentation explaining the DARK ROOM server installation, configuration, and deployment steps.
• Committed all documentation files under a clear commit message.

7.3.5 3. Submission

• Pushed the new branch to my GitHub fork.
• Created a pull request titled **"DARKROOM Self Hosted Project Documentation Added"**.
• Provided a clear PR description stating that the documentation covers the complete DARK ROOM setup process.

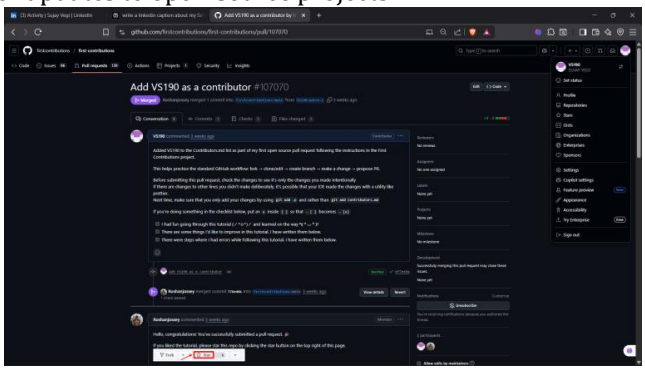
• Ensured all commits were clean and ready for review.
• GitHub confirmed **"No conflicts with base branch"**, meaning the PR could be merged safely.

7.3.6 Difficulties and Solutions

The main challenge was ensuring the documentation was complete, accurate, and easy to follow.
I solved this by testing each step on my hosted server and documenting the process in a structured and clear manner.
No merge conflicts or technical issues occurred due to clean branching.

7.3.7 Next Steps

This PR is currently open and ready for maintainer review.
Once merged, it will help future students and contributors understand how to self-host DARK ROOM easily.
This contribution strengthened my documentation skills and improved my confidence in contributing larger updates to open-source projects.



7.4 PR 4 : Typo Fix in SCP Command Documentation

7.4.1 Goal

The goal of this pull request was to correct a spelling mistake in the **SCP command documentation**.
The file contained the word **"though"** instead of **"through,"** which changed the meaning of the sentence.
The purpose was to improve accuracy and clarity for users learning Linux commands.

7.4.2 The Contribution Workflow

This contribution followed the typical workflow:
fork → clone → branch → edit → commit → push → pull request.
It helped me understand how even small documentation fixes are handled in open-source projects.

7.4.3 1. Setup

• **Fork:** I forked the repository into my GitHub account.
• **Clone:** Cloned the repository locally using Git.
• **Preparation:** Identified the exact file and line where the typo existed.

7.4.4 2. Making Changes

• Created a new branch for the fix.
• Located the typo in the file ebook/en/content/076-the-scp-command.md (line 30).
• Corrected the text from **“To copy file though a jump host server”** to
“To copy file through a jump host server.”
• Committed the update with a descriptive message.

7.4.5 3. Submission

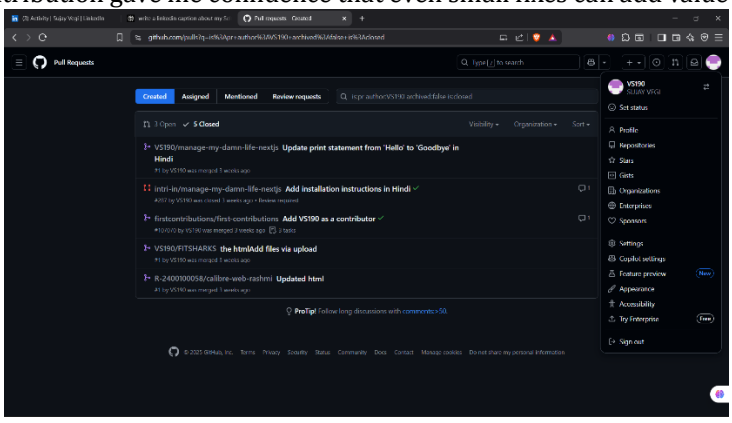
• Pushed the branch to GitHub using git push.
• Opened a pull request titled
“Fix typo: ‘though’ to ‘through’ in scp command.”
• Filled out the PR checklist confirming:
– I followed the contributing guide
– The commit message is descriptive
– Only one small, focused change was made
• Submitted the PR for review.

7.4.6 Difficulties and Solutions

There were no major technical difficulties.
The small challenge was ensuring the commit followed project rules and that only the intended line was changed.
Reading the contributing guide helped avoid mistakes and kept the PR clean.

7.4.7 Next Steps

The pull request is currently open and waiting for approval.
Once merged, the documentation will be clearer for users learning the SCP command.
This contribution gave me confidence that even small fixes can add value to large open-source projects.



8 Linkedin Post Links

8.1 PR :

https://www.linkedin.com/posts/sujay-vegi-5090b1204_opensource-kl-hte-activity-7399418331528691712-OovH?utm_source=social_share_send&utm_medium=member_desktop_web&rcm=ACoAADP7xJkBb4shsVJEQ8IvemWRzKLkK0HEGrQ

Journey Of Open Source :

https://www.linkedin.com/posts/sujay-vegi-5090b1204_activity-7399385363837972480-kna2?utm_source=social_share_send&utm_medium=member_desktop_web&rcm=ACoAADP7xJkBb4shsVJEQ8IvemWRzKLkK0HEGrQ

Self Hosted Project :

https://www.linkedin.com/pulse/my-linux-open-source-hosting-journey-kl-university-sujay-vegi-rlswf

https://www.linkedin.com/posts/sujay-vegi-5090b1204_opensource-klusamyak-calibreweb-activity-7383369122174291969-rDLU?utm_source=social_share_send&utm_medium=member_desktop_web&rcm=ACoAADP7xJkBb4shsVJEQ8IvemWRzKLkK0HEGrQ