

# Raspberry Pi & Linux Guide

## I - What is a Raspberry Pi and Linux?

### Raspberry Pi

#### What is it?

[The Raspberry Pi](#) is a series of small, affordable, single-board computers developed in the United Kingdom by the Raspberry Pi Foundation in collaboration with Broadcom. It's designed to promote teaching basic computer science and to provide an accessible platform for hobbyists, educators, and students to explore computing, learn programming, and work on various digital projects.

#### Raspberry Pi Projects

[Raspberry Pi projects](#) range from simple learning tools to complex systems. Some popular project ideas include:

- [Coding and Computing](#): Using Raspberry Pi to learn programming languages like Python and Scratch.
- Home Automation: Creating smart home devices that can control lighting, temperature, or security systems.
- Retro Gaming: Building a retro gaming console to play classic games.
- [Robotics](#): Designing robots that can perform various tasks or respond to the environment.
- Media Centers: Setting up a media server to stream videos and music throughout your home.

#### Raspberry Pi Documentation

To read more about the Raspberry Pi, you can visit the official Raspberry Pi website or explore its Wikipedia page for detailed information on its history, models, and specifications. Additionally, there are numerous online communities and forums where Raspberry Pi enthusiasts share their projects and offer support.

## Linux

#### What is Linux?

- Linux is an open-source operating system (OS) that has been around since the mid-1990s. It's part of the Unix-like family of OSes.
- Unlike proprietary systems like Windows or macOS, Linux is freely redistributable, which means anyone can create their own distribution (distro) for any purpose.
- The core of Linux is the Linux kernel, which manages the CPU, memory, and peripheral devices. However, when people refer to "Linux," they often mean the entire OS, including

additional software and tools.

- Linux is everywhere: from smartphones and cars to supercomputers and home appliances. It powers most of the Internet, top supercomputers, and stock exchanges.
- Android, one of the most popular platforms globally, is also based on the Linux kernel.

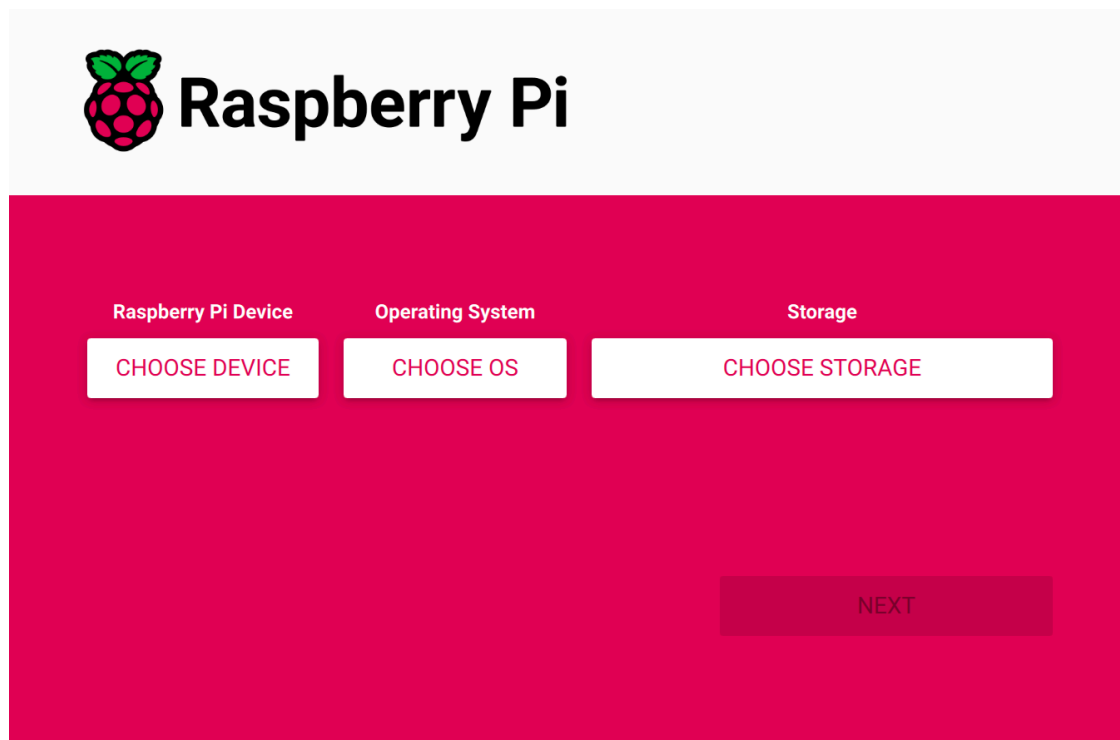
### Why does the Pi use Linux?

- The Raspberry Pi Foundation has a unique mission: to help young people learn how to use computers and improve their tech skills.
- They created the Raspberry Pi, an affordable computer (costing less than \$50) that can be used in schools. It helps students become more confident with computers and learn programming.
- However, the Raspberry Pi is different from traditional computers, so a specialized OS was needed. That's where Raspberry Pi OS (formerly Raspbian) comes in.
- Raspberry Pi OS is a lightweight Linux distribution designed specifically for Raspberry Pi. It achieves the foundation's goals by providing a compatible and user-friendly environment for learning and development.

## II - How to use Linux

### Installation

Install the Raspberry Pi Imager on a computer. This is used to install a .iso file to an SD card or MicroSD card (depending on your Raspberry Pi model). Launch the application and you'll see the following buttons:



Select your Raspberry Pi device and the storage device connected to your computer.

For the Operating System, if you have a Raspberry Pi 3 or newer, the top option should be OK. Anything older, and you should choose Raspberry Pi OS (Legacy, 32-bit) Lite. Note that this will not install a Desktop Environment (DE), which means everything you'll do is through the terminal. That might sound scary, but don't worry. This guide will go over everything you need to know to hit the ground running.

Once you hit next, you should see the popup below:

GENERAL

SERVICES

OPTIONS

☐ Set hostname: .local

☒ Set username and password

Username:

Password:

☐ Configure wireless LAN

SSID:

Password:

☒ Show password ☐ Hidden SSID

Wireless LAN country:

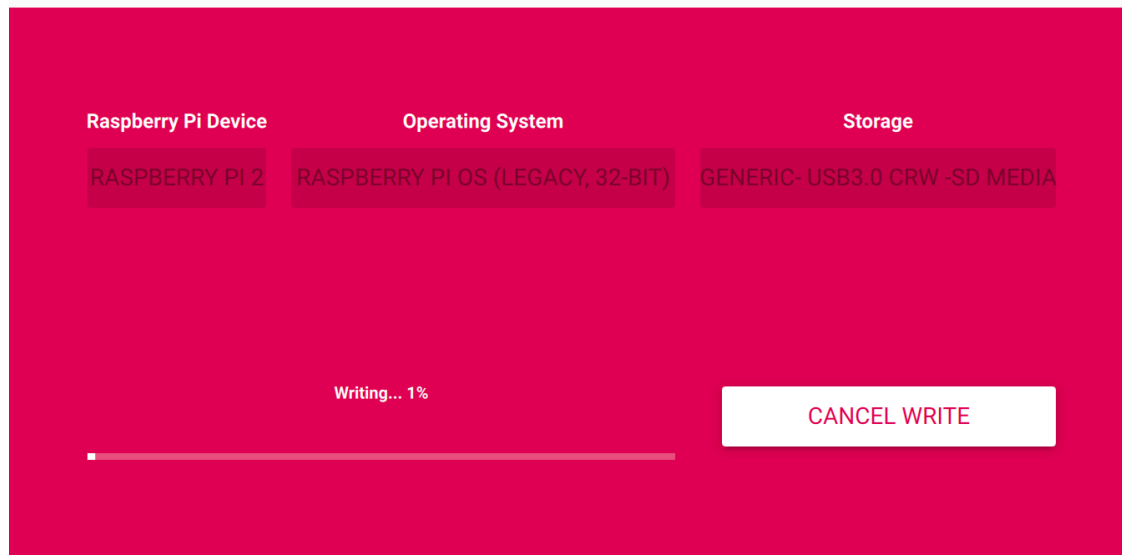
☐ Set locale settings

Time zone:

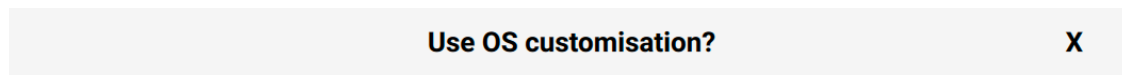
Keyboard layout:

SAVE

Click **EDIT SETTINGS** and **Set username and password**. This is the only setting you *should* set, but others are available, such as enabling SSH (Secure Shell protocol). For the time being, let's skip those since they're not strictly necessary.



Now, wait until the installation is complete!



Would you like to apply OS customisation settings?



## SSH vs. Local

Using a Raspberry Pi with Linux can be experienced in two distinct ways: through SSH (Secure Shell) or locally by connecting a display directly to the Pi. Here's a comparison of both methods:

### Using SSH:

- [Remote Access](#): SSH allows you to control your Raspberry Pi from another computer on the same network or even over the internet.
- [Headless Operation](#): Ideal for running the Raspberry Pi without a monitor, keyboard, or mouse—useful for servers or embedded projects.
- [Convenience](#): Manage your Pi from the comfort of your main workstation, which is especially handy if the Pi is physically inaccessible.

- [Development](#): Tools like VS Code can connect to the Pi via SSH, enabling you to write and execute code remotely.
- [Security](#): SSH provides encrypted communication, ensuring that your commands and data are secure during transmission.

### Using Locally:

- Full Desktop Experience: When a display is connected, you can use the Raspberry Pi's graphical user interface (GUI) for a complete desktop experience.
- Peripheral Interaction: Directly interact with connected devices such as keyboards, mice, or other USB peripherals.
- Educational Use: For learning and teaching, having a monitor connected can be more engaging and easier for beginners.
- No Network Required: Local use doesn't depend on network connectivity, which can be beneficial in environments with poor or no network access.

If you are going to utilize SSH, please be sure to check that option in the settings page when you are installing, though this can be enabled in the future.

## III - The Power of the Linux Terminal

### Why use a terminal?

A command is just an application. Some commands open a window (a Graphical User Interface or GUI) while others remain in the terminal (referred to as Command-Line Interface or CLI). Many developers create applications that live in the terminal because, frankly, it's a lot less work. What this means is that small, specialized tools are widely available on Linux since someone likely has already made it.

While there are many, many more commands, this guide will stick to the essentials to get you up and running quickly. The essentials have been divided into the following categories:

- File Operations
- Permissions
- Installing Applications
- Coding
- Git Essentials
- Miscellaneous

These commands can be accessed in the terminal and can have many different options (called parameters outside of a program, arguments inside a program). Google is your friend here, but an even better friend is `tl;dr`. It provides simple and concise explanations of commands with examples for the most common uses of those commands.

Learning these commands will take time. It will at times be frustrating. Like anything worthwhile, it takes time, practice, and patience.

# Essential Terminal Commands

Most commands have different parameters, some optional and some required. This guide will only cover the most common uses of the most common commands. Use `tlldr` command to get more uses for a command. Use `man` command to get detailed information and all parameters for a command.

For most commands, either a full path or relative path may be used for input/output files and directories. Make sure to consider your user access to paths as well. For a command with input and/or output parameters, the input must be a file/directory that your user has access to read. The output must be a file/directory that your user has access to write to.

Most commands also support wildcard characters. Once you're familiar with commands, start experimenting with wildcards. Start by reading more [here](#).

## File Operations

### cd

`cd` is by far the most commonly used command line utility. This changes what directory you are currently in.

Command	Description
<code>cd</code>	Go to your home directory, <code>/home/username</code> .
<code>cd ~</code>	Go to your home directory, <code>/home/username</code> .
<code>cd /home/username/directory</code>	Go to a specific directory by providing a full path.
<code>cd directory</code>	Go to a directory that is in your current directory.

### ls

`ls` is the second most used command. This command will list files and directories in your current location by default, but there are multiple options.

Command	Description
<code>ls</code>	Lists files and directories in your current directory.
<code>ls -l</code>	Include more information like ownership and permissions.
<code>ls -la</code>	Include hidden files and directories.
<code>ls -la directory</code>	List all files and information for a specific directory.

### mkdir

`mkdir` is used to create a new directory or folder, or many directories at once.

Command	Description
<code>mkdir a-directory</code>	Create a new directory.
<code>mkdir a-directory b-directory ...</code>	Make multiple directories.

## **touch**

`touch` has multiple uses. The primary use is to create new file(s). The secondary use is to update the last-accessed timestamp of a file. Note that this command does not create directories, but will update the timestamp of a directory.

Command	Description
<code>touch file.txt</code>	Create (or update the access timestamp) a file.
<code>touch file.txt a-directory ...</code>	Create/update a file and update a directory.

## **cp**

`cp` copies a file(s)/directory(ies) to a destination directory. This copies the permissions as well as the file contents themselves. `cp` writes over any existing output file as well.

Command	Description
<code>cp input-file output-file</code>	Copy the input file and create/write the output file.
<code>cp a-input b-input directory</code>	Copy multiple files into one output directory.
<code>cp -r input-directory output-directory</code>	Create a copy of the input directory with the output's name.

## **mv**

`mv` moves a file(s)/directory(ies) to a target destination. This can be used to rename a file or directory, or move file(s)/directory(ies) into another location.

Command	Description
<code>mv input-file directory</code>	Move the input file into the directory.
<code>mv a-input b-input directory</code>	Move multiple files into one directory.
<code>mv input-name output-name</code>	Rename a directory/file.

## rm

`rm` is the delete command, and can remove one or more files at once, and can be used to delete directories as well.

Command	Description
<code>rm input-file</code>	Remove a file.
<code>rm a-input b-input</code>	Remove multiple files at once.
<code>rm -r a-directory</code>	Remove a directory.

# Permissions

## sudo

**sudo** allows a command to be run as the root user (see II - How to use Linux). This is most commonly used in conjunction with your package manager (see Installing Applications), but there are many use cases where you'll want to use it. From editing system configuration files to executing system commands, in Windows terms `sudo` elevates your command to be run as the Administrator.

Related: `su`

Association: **switch user** and **do** || **super user** and **do**

Command	Description
<code>sudo nvim /usr/local/bin/a-custom-script</code>	Since a regular user doesn't have permission to edit a file in <code>/usr/local/bin</code> , switch to the root user to edit the file.
<code>sudo apt install firefox</code>	Installing, updating, upgrading, and removing packages requires <code>sudo</code> (see Apps).
<code>sudo cp new-script /usr/local/bin</code>	Copy a new script to a globally accessible script location (see IV - Making your own CLI tools).

## Apps

## Package Manager == App Store

A package manager is an app store (in coding, the '==' operation is a boolean expression meaning the left and right are the same value). There are a few important differences though.



- A package manager is “distro-dependent”, meaning it is only available to and is designed for specific Linux distributions and families. A package manager can also be available and designed for use for a specific programming language as well (see Setting Up Programming Languages).
- A package manager is usually a **Command Line Interface**, meaning that your interaction with it is limited to the terminal.
- If an application is already installed and you attempt to re-install it, the exact behavior can vary but either the package manager aborts the operation or it re-installs the app.

## Raspberry Pi

For the Raspberry Pi, the package manager is **Advanced Package Tool**, commonly called `apt`. Packages are applications, with all dependencies and instructions already set up.

The `install` and `upgrade` sub-commands will typically require some input from you. This comes in the form of simple `Y` or `N` prompts typically.

Command	Description
<code>sudo apt update</code>	Check if any packages have a new version available.
<code>sudo apt install firefox</code>	Install the Firefox web browser.
<code>sudo apt remove firefox</code>	Uninstall the Firefox web browser.
<code>sudo apt upgrade</code>	Upgrade every package installed with <code>apt</code> to the latest version.
<code>apt search firefox</code>	Find a package based upon search criteria.

# Coding

Linux is a programmer’s playground. Writing code that will modify your system, create applications, and CLI tools, are all easily accessible through Linux. This section will cover how you can write code in the terminal, and how you can run that code.

## Text Editor

### NeoVim

High-level, Neovim is a customizable and extensible text editor. You can either install a pre-configured Neovim distro (like [LunarVim](#), [AstroNvim](#), or [LazyVim](#)) or start from scratch. While you’re learning, a pre-configured distro is going to smooth the learning curve and is the general recommendation for starting out with Neovim. It’ll include modern conveniences like auto-complete, GitHub Copilot, warnings and errors, and more.

Eventually, creating a custom configuration is ideal but ultimately not necessary. The keybindings below are the same across distributions. Additionally, most servers will have at least Vi (an ancestor of Neovim) installed. Part of the power of Neovim (and its ancestors Vi and Vim) is the

modal interface. Fancy words aside, this just means that you only operate within different modes. In different modes, the keybindings change. This will only cover the most common and necessary keybinds and commands.

Command	Description	Category
<code>nvim</code>	Open a new file. The file is created with its contents once you save.	Basic file creation
<code>nvim file.txt</code>	Open a file called <code>file.txt</code> .	Open a file
<code>nvim .</code>	Open the current directory in Neovim, using <code>netrw</code> .	Enter <code>netrw</code>
<code>nvim a-directory</code>	Open a directory from a path, using <code>netrw</code> .	Enter <code>netrw</code>

### Keybinds:

Part of the power of Neovim is repeatable commands. The commands below can have a number prefix to indicate how many times to perform that command. Eg. `50j` in command mode translates to moving 50 lines down.

Keybind	Description	Category
<code>h</code>	Move the cursor left a column, but in the same row.	Navigation
<code>j</code>	Move the cursor down a row, and attempt to preserve the column.	Navigation
<code>k</code>	Move the cursor right a column, but in the same row.	Navigation
<code>l</code>	Move the cursor up a row, and attempt to preserve the column.	Navigation
<code>w</code>	Move the cursor to the start of the next word or special character, ignoring whitespace.	Navigation
<code>b</code>	Move the cursor to the start of the previous word or special character, ignoring whitespace.	Navigation
<code>i</code>	Enter <code>insert</code> mode, the primary typing mode.	Insert Mode
<code>I</code>	Enter <code>insert</code> mode at the start of the current line.	Insert Mode
<code>S</code>	Enter <code>insert</code> mode after clearing the current line.	Insert Mode
<code>A</code>	Enter <code>insert</code> mode at the end of the line.	Insert Mode

v	Enter <code>visual</code> mode (highlighting/selecting) at the cursor's current location. Select text with navigation keys.	Visual Mode
V	Enter <code>visual</code> mode at the cursor's current row. <code>j</code> and <code>k</code> are used to select more rows.	Visual Mode
y	Copy highlighted text to Neovim's clipboard. <b>NOTE:</b> Neovim's clipboard is separate from your system clipboard.	Visual Mode
"*y	Copy highlighted text to your system clipboard.	Visual Mode
u	Undo.	Special Function
p	Paste text from Neovim's clipboard.	Special Function
:Ex	Enter Neovim's file explorer, <code>netrw</code> .	Enter <code>netrw</code>
%	Create a file inside the current directory.	<code>netrw</code> keybinds
d	Create a new directory inside the current directory.	<code>netrw</code> keybinds
D	Delete a file or empty directory.	<code>netrw</code> keybinds
:w {filename}	Write (save) the current file. You can specify a filename as well.	File Management
:q	Quit (exit without saving) the current file.	File Management
:wq	Write (save) and quit (exit) the current file.	File Management
:%s/original/new/g	The most common command I use, a highly and easily configurable search and replace. The (optional) <code>%</code> means the entire file. By default, it'll only apply to the current line. <code>original</code> is the current text and <code>new</code> is the replacement. The (optional) <code>g</code> means every instance on each line. By default, it'll only apply to the first occurrence.	Advanced Command
Escape	Exit current mode.	Exit mode

# Setting Up Programming Languages

A broad overview on a handful of languages that you can use with the Raspberry Pi:

Python is the most commonly used language with the Raspberry Pi. Python is known for its syntactic simplicity for novice programmers...

## Python

1. Run `sudo apt install python`.
2. Verify the installation was successful with `python --version`.
3. Verify that the python package manager was installed with `pip --version`.

# Git Essentials

Git is a critical component of modern software development...

## git clone

Command	Description
<code>git clone https://github.com/&lt;user&gt;/&lt;repo&gt;</code>	Create a local copy of the remote repo owned by <user> called <repo>.
<code>git clone &lt;remote&gt; &lt;destination&gt;</code>	Clone a remote repository and save it in a directory called <destination>.

## git stage

Command	Description
<code>git stage -A</code>	Include all changes in the commit.
<code>git stage &lt;file&gt;</code>	Add a file you want to include in the commit.

## git commit

Command	Description
<code>git commit</code>	Open your default text editor to create the commit message. Once you save and quit, a new snapshot of your local repo is saved.

<code>git commit -m "A commit message."</code>	Create the snapshot with a commit message.
<code>git commit --all</code>	This option auto-stages your changes for the commit, so you do not need <code>git stage</code> .

## git push

**git push** will attempt to upload your commit(s) to the remote repository.

## git pull

**git pull** will download the latest changes from the remote repository.

# Quality of Life Improvements

## man

**man** is used to open instructional information about a particular command, application, or function. The `man` command is quite powerful, though a touch verbose. It'll provide information on all possible parameters, uses, and idiosyncrasies.

Command	Description
<code>man &lt;command&gt;</code>	Open the manual for a command, application, or function. Press <code>q</code> to quit, and use <code>j</code> and <code>k</code> to move down and up respectively.

## tldr

**tldr** is the concise version of `man`. Personally, I use `tldr` almost every day. It provides the most common few uses of a command, application, or function.

Command	Description
<code>tldr &lt;command&gt;</code>	Get quick information about a command, application, or function.

## zsh

**zsh** is a shell, effectively the programming language your terminal uses. `zsh` has more customization options available, such as `oh-my-zsh`, `starship`, and much more. Using your programming skills to customize your operating system is a deep rabbit-hole that you should at least be aware of.

# IV - How to unleash the Raspberry Pi's Potential.

Here is an example script (found [here](#)) which gets your Raspberry Pi setup with some common services.

```
#!/bin/bash

# This script gets a Raspberry Pi setup with essential applications.
#
# For explanations of commands, visit raspi.vintagecoding.net

# GLOBAL VARIABLES
DOMAIN=""
MANUAL_PORT=5090
JELLYFIN_PORT=5091
OLLAMA_PORT=5092

# This function initializes core services and packages for getting started.
init() {
    # Update repositories and upgrade installed packages.
    sudo apt update && sudo apt upgrade;

    # Install requisite packages for cloudflared.
    sudo apt install curl lsb-release;

    # Add cloudflare gpg key
    sudo mkdir -p --mode=0755 /usr/share/keyrings
    curl -fsSL https://pkg.cloudflare.com/cloudflare-main.gpg | sudo tee
/usr/share/keyrings/cloudflare-main.gpg > /dev/null

    # Add this repo to your apt repositories
    echo 'deb [signed-by=/usr/share/keyrings/cloudflare-main.gpg]
https://pkg.cloudflare.com/cloudflared any main' | sudo tee
/etc/apt/sources.list.d/cloudflared.list

    # install cloudflared
    sudo apt-get update && sudo apt-get install cloudflared

    # Install cloudflared.
    # Install podman, a container technology for easily deploying projects.
    # Install golang, a server-side programming language.
    # Install tools for the user
    sudo apt update;
    sudo apt install cloudflared podman golang tldr neovim;

    # Get user input for some default applications to get started.
    read -p "Do you want a mirror of the manual from raspi.vintagecoding.net
on this device? (y/N): " MANUAL;
    MANUAL=$(echo "$MANUAL" | tr '[:upper:]' '[:lower:]')
    if [[ -z "$MANUAL" || "$MANUAL" == "n" ]]; then
```

```

        MANUAL="n"
    else
        init_manual;
    fi

    read -p "Do you want to setup Jellyfin, a personal Netflix alternative?
(y/N): " JELLYFIN;
    JELLYFIN=$(echo "$JELLYFIN" | tr '[:upper:]' '[:lower:]')
    if [[ -z "$JELLYFIN" || "$JELLYFIN" == "n" ]]; then
        JELLYFIN="n"
    else
        init_jellyfin;
    fi

    read -p "Do you want to setup Ollama, a personal ChatGPT alternative?
(y/N): " OLLAMA;
    OLLAMA=$(echo "$OLLAMA" | tr '[:upper:]' '[:lower:]')
    if [[ -z "$OLLAMA" || "$OLLAMA" == "n" ]]; then
        OLLAMA="n"
    else
        init_ollama;
    fi

}

# This function initializes the manual mirror of raspi.vintagecoding.net
init_manual() {
    cd ~/ras-pi/man/pages;

    # This creates what's called a systemd service. These are processes that
    you create which can start after a reboot.
    echo "[Unit]
Description=Raspberry Pi Manual Mirror
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=always
RestartSec=1
User=$(whoami)
WorkingDirectory=$(pwd)
ExecStartPre=/usr/bin/git pull
ExecStart=/usr/bin/python3 -m http.server $MANUAL_PORT

[Install]
WantedBy=multi-user.target" | sudo tee
/etc/systemd/system/manual.service;

    sudo systemctl enable manual.service;
    sudo systemctl start manual.service;
    sudo systemctl daemon-reload;
}

```

```

# This function initializes Jellyfin.
init_jellyfin() {
    # Get the official image of Jellyfin software
    podman pull ghcr.io/jellyfin/jellyfin;

    echo "Making media directories where Jellyfin will retrieve content.";
    echo "It is up to you to provide supported files and build a library.";

    cd;
    mkdir media;

    podman run \
        --detach \
        --label "io.containers.autoupdate=registry" \
        --name myjellyfin \
        --publish $JELLYFIN_PORT:8096/tcp \
        --user $(id -u):$(id -g) \
        --volume jellyfin-cache:/cache:Z \
        --volume jellyfin-config:/config:Z \
        --mount
type=bind,source=/HOME/media,destination=/media,ro=true,relabel=private \
        --restart always \
        ghcr.io/jellyfin/jellyfin;

    echo "Finished creating the Jellyfin container! It is now live on";
    echo "http://localhost:$JELLYFIN_PORT. If you enabled it, it'll also
be";

    echo "available at media.yourdomain.com.";
}

# This function initializes Ollama.
init_ollama() {
    echo "This will take a while...";

    cd;
    curl -fsSL https://ollama.com/install.sh | sh;
    export OLLAMA_BASE_URL=http://localhost:11434;

    read -p "Do you want to require a login for your AI? (n/Y): "
REQUIRE_LOGIN;
    REQUIRE_LOGIN=$(echo "$REQUIRE_LOGIN" | tr '[:upper:]' '[:lower:]')
    if [[ -z "$REQUIRE_LOGIN" || "$REQUIRE_LOGIN" == "n" ]]; then
        export WEBUI_AUTH=False;
    fi

    source $HOME/.bashrc

    # TODO: Write a script so this can be daemonized.
    python -m venv open-webui-env;
    source open-webui-env/bin/activate
    pip install open-webui;

    echo "server {
        listen 80; #or whatever port your open-webui backend is running on.

```



```

server_name localhost;

location / {
    proxy_pass http://localhost:$OLLAMA_PORT/;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_cache_bypass $http_upgrade;
}
location /api/ {
    proxy_pass http://localhost:$OLLAMA_PORT/api/; # Ensure the
trailing slash is present
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_cache_bypass $http_upgrade;
}
}" | sudo tee /etc/nginx/sites-available/open-webui;
sudo ln -s /etc/nginx/sites-available/open-webui /etc/nginx/sites-
enabled

sudo systemctl enable nginx;
sudo systemctl start nginx;

open-webui serve --port $OLLAMA_PORT &
}

init;

```