Creating a Linux Home Server

Denice Deatrich

Created: March 2023 Last update: April 22, 2023

Contents

Overview

Single-board computers (SBC) are both inexpensive and reliable; they are also very small. As such they make excellent 24x7 home servers. This guide steps you through the process of creating such a server.

This guide has been tested on a Raspberry Pi 400, which is very similar to a Raspberry Pi 4b. The main difference it that the RP 400 board is embedded in a small keyboard.

I also have another SBC, an ODROID, and I will document it as well. I am not sure yet whether I will document it here, or as another mini-document. For now I am focusing on the Raspberry Pi.

One of my goals is to promote using the command-line to do most of the work. If you are interested in expanding your horizons and understanding more about the command-line, then this guide is for you.

Take a look at the table of contents at the top of the document. Some users will only be interested in creating a 24x7 local file-sharing (Samba) service at home – in that case you do not need to read beyond the section on 'Backups'. Of course, people familiar with Linux and the command-line will skip some sections of this guide.

A Few Issues Before You Begin

You should look into assigning a permanent network IP address for your server in your home network so that you can easily connect to it from any of your devices. Your home network router/WiFi modem should have the option to enable you to reserve an IP address for any device. You only need to know the hardware MAC address of your future server. There will be 2 MAC addresses - one for hardwired ethernet and one for wireless. You can reserve both interfaces until you decide which way you will connect your server to your router.

About this Document

This guide was created in the *Markdown* markup language (the Pandoc flavour of markdown). Markdown is wonderfully simple. Then, using the versatile Pandoc command set, both HTML and PDF formats of the document were generated. In fact this document was created using the home server as a remote desktop. The server served as a git, web and NFS server; as well it served as a remote desktop for contemporary documentation creation.

The appendix of this document is rather large. The idea is to push some of the command-line and technical detail into the appendix. Thus the flow of the document covers the basics, encouraging the reader to see the bigger picture and to avoid being smothered in the detail.

In this guide command-line sessions show two kinds of simplified command-line prompts:

Normal Users:
$

The root superuser:
#

Also, command output is sometimes long and/or uninteresting in the context of this guide. I might show such segments with a ellipsis (**...**)

If you discover issues with instructions in this document, or have other comments or suggestions then you can contact me on github in my documentation projects area.

Picking the OS (Operating System) and the Window Manager

Though many Raspberry Pi owners run the Raspberry Pi OS (Raspian), in this guide I chose to use Ubuntu. Ubuntu LTS, is a long-term support Debian-based Linux OS. Ubuntu is renowned for its desktop support, but it also provides a comfortable home server experience. A server should be stable. We want to apply software updates, but we also want to avoid the need to update the major version of the base OS every year. The Official Gnome LTS releases with the Gnome desktop environment in the *main* software repository are supported for up to 5 years from initial release, and up to 10 years with extended security-only updates via Ubuntu Advantage access[1]. However if you do not have an *advantage* account, then community-supported desktop environments which are located in the *universe* software repository only get 3 years of support, meaning system-wide there is only partial support after 3 years. Nonetheless, many critical services are installed from the base repository, and they have the usual 5 years of support.

Generally you should think about updating your server OS every few years so that you stay in touch with current technologies.

[1] Ubuntu Advantage is also known as **Ubuntu Pro**. It is free of charge for personal use on up to 5 machines.

At the time of writing this guide I used version 22.04 of Ubuntu LTS (also known as **Jammy Jellyfish**). It was first released in April 2022.

I also opt to use an installation image which uses the MATE desktop system. At the bottom of that linked website is a note about why it is called MATE (pronounced mat-ay). The MATE window manager is intuitive, efficient, skinny, dependable and popular. It is widely available on most flavours of Linux.

Even though we are creating a home server, it is useful to configure the server to provide a remote graphical desktop environment – this is why in this guide we use the desktop image rather than the server image. Then you can use the desktop for fun, learning, or perhaps as your Linux development environment from other devices. Accessing the desktop remotely is also documented in this guide.

Creating the Installation Disk

You will need a new or repurposed microSD card with a capacity of at least 32 GB – since this is a server we might want to store lots of photos or videos on it. See the Ubuntu MATE website for some examples of microSD cards. Recently I was able to buy a 256 GB Silicon Power microSD card for less than $25 Cdn on Amazon; this brand is rated highly for use on Raspberry Pi's by testers like Tom's Hardware.

Go to the Ubuntu MATE download website to download your image - for a Pi 4 generation with 4 or more GB of RAM the 64-bit ARM architecture (arm64) is best. For the version I used in March 2023 the image name was: *ubuntu-mate-22.04-desktop-arm64+raspi.img.xz*.

There are many instructions available online to help you download the disk image and install it onto installation media - I will not reproduce the instructions here. How you create the image depends on your home computing device and its OS. There is a helpful Ubuntu tutorial on creating the installation image using the Raspberry Pi Imager software for 3 operating systems:

- Windows OS
- MacOS
- Debian-based Linux

Note that after installing the disk image on the microSD the disk partitioning looks like this. I only show it here so that you are aware of what is going on under the hood. Here is an example of a 256 GB microSD inserted into a USB card reader on another Linux computer where the card showed up as /dev/sde:

```
# fdisk -l /dev/sde
Disk /dev/sde: 231.68 GiB, 248765218816 bytes, 485869568 sectors
Disk model: FCR-HS3       -3
...
Disklabel type: dos
Disk identifier: 0x11d94b9e

Device     Boot  Start      End  Sectors  Size Id Type
/dev/sde1  *      2048   499711   497664  243M  c W95 FAT32 (LBA)
/dev/sde2       499712 12969983 12470272  5.9G 83 Linux
```

So there are 2 partitions; the first (/dev/sde1) is a small boot partition whose type is FAT32, and the second (/dev/sde2) is the minimal 6 GB Linux partition. Though this microSD is 256 GB only the first 6 GB is currently used. The automatic installation process will expand the partition right to the maximum extend of its partition or of unallocated space. Most Linux installation images allow you to choose your disk partitioning; the Raspberry Pi installation image does not.

However, it is possible and useful to modify the pre-installation partitioning directly on the microSD card as described in the appendix.

In the appendix I also provide a generic Linux command-line approach to downloading, uncompressing and writing the image to the microSD card. If you are not yet very familiar with the command-line then leave this exercise for a later time in your Linux adventure.

Installation and First Experience

Once you have prepared your microSD card then insert it in your Raspberry Pi. Note that the card pushes in easily. It will only go in one way. To eject it gently push in on it once and it will pop out enough to handle it. There is no need to pull on it to remove it because it essentially pops out.

Turn the power on with the Pi connected to a monitor, USB keyboard and mouse. You will shortly see the firmware rainbow splash screen. Shortly after that there are a series of screens allowing you to customize the installation:

- pick your system language
- pick the keyboard layout language
- enable the Wi-Fi network access if you want to have concurrent updates
- select your timezone by clicking on your timezone region
- enter your preferred name and your login name - this is your login account

As the installation starts it will show some informational screens to entertain you while it installs. Eventually it will reboot and present you with the login screen. Once you login you will see the default MATE desktop configuration.

Experimenting With the Graphical Environment

If you are not overly familiar with Linux environments then this is a good time to take a few days and play with the installation while it presents a graphical interface. Note that in the Server Customization chapter we will *remove* the graphical login screen. This will not stop you from using the Linux desktop environment remotely. As well, you can still invoke on demand a graphical presentation locally by logging into the text console and running the *startx* command.

Remember that you can always reinstall the server by re-imaging the microSD card to start over. It does not take much time.

If you are not comfortable with the command-line, why not take a few days and play with the login environment and experiment with the settings? There is a MATE exercise in the appendix that you can try.

Some Linux Basics

Here are a few general but important concepts to review for people less familiar with Linux.

Privileges The privileged account on a Linux system is the superuser whose login name is 'root'. You should never log into a graphical environment as root. When you install a desktop Linux system you create an account for yourself. Your account does not have special privileges except when you deliberately invoke them.

As a regular user you can create and manage files in your home folder, which is normally in the /home directory. There are a few other special directories where you can create and manage files, but only your files. Those directories are */tmp* and */var/tmp*

All other directories and most files are off-limits. The root user manages the system, including device changes, system software updates, and management of all services.

The usual way to become temporarily the root user is to use the *sudo* command to run other commands on behalf of the root user. When you use the sudo command you will be asked for your password to verify your identity [2]. As well, all the commands you run under sudo are logged (on Ubuntu they are logged in */var/log/auth.log*).

System Management Basics Here are a informational links about a couple of system management tools used in this guide:

- Linux Package Management (for installing, updating and removing software)
- Systemd Tools (for an explanation about task management with systemctl)

[2] It is possible to add a *timestamp_timeout* factor in minutes to the */etc/sudoers* file to allow you to run other sudo commands for some minutes without entering your password.

Directory Structure In Linux and UNIX based systems the file system begins at the symbol known as a forward slash: */*

Here is a table of the typical major directories and their purpose. There are also some top level directories like */run*, */proc* and */sys* which house virtual files for access to kernel and memory resident runtime data.

Directory	Purpose
/	The base of the file system
/boot	The kernel and the bios and firmware utilities live here
/etc	Generally system configuration files are here
/home	Regular users will find their home directories here
/mnt	You might mount a foreign file system inside here
/root	The home directory for the superuser
/usr	Here we have programs, shared files, documentation, system libraries and programming files
/var	Varying files are here - log files, cache files, database files, web files, and others (like databases)
/dev	Special device files reside here

Editing Text Configuration Files The classic UNIX editor for system administrators is *vi*. The associated Linux version is *vim*. If you do not know vim then you should consider learning it. Try the program named *vimtutor* to self-teach yourself about vim. When vim's graphical version, *gvim* is installed then try *gvimtutor*.

There are some useful simple text editors that work without graphics. A popular editor named *nano* is the best place to start if you don't want to try text editors with a learning curve. When you invoke nano immediately enter ^G (meaning hold the CONTROL key while pressing the **g** key) to get some help.

Getting Going with the Command-Line

For people without much command-line experience it is important to get going at the command-line. When logged into the MATE desktop open a terminal window by selecting Application -> System Tools -> MATE Terminal.

Try some command-line examples in the appendix. Note that the up/down arrow keys can be used to recall your previous commands. You can edit an entry in your previous commands using the left/right arrow keys.

You will find that the 'TAB' key (shown below as) is very useful for command-line completion. Suppose you are going to use the command 'timedatectl'. You start by typing the word 'time' and then hit the TAB key once, then again when you do not get a response. You will see 4 possible commands as shown below. Then to complete the command simply type d followed by another TAB and the full command will complete:

```
$ time<TAB><TAB>
time          timedatectl  timeout      times
$ timed<TAB>
$ timedatectl
```

Look online for some tutorials; there are millions of results on Google if you search for:

```
Linux "command line" tutorial for beginners
```

Creating a Samba File Sharing Service

We are going to create a Samba file sharing service on our server. Other devices like mobile phones, tablets, laptops and desktops running a variety of operating systems should be able to manage files in the designated data area.

We are not going to be really secure, in that we are allowing guest access. Presumably if you let your family and your guests connect to your network, then you would allow them to connect to your Samba server.

But as always, your internal home network should be protected with at least a strong password for your wireless SSID connections.

Create the data space and assign appropriate permissions.

First, visit Setting Up a Data Area in the appendix so that you create a data area.

Always use a sub-directory inside the data area to begin any new project. One of the advantages is that the lost+found directory does not become part of your project. For this Samba project we will create */data/shared*.

For Samba the top level ownership of the samba area will be a user named 'nobody'. This user is always created in Linux systems and has no login shell, so 'nobody' cannot log in. It is a safer user identity to use for guest access to Samba shares.

We set the access permissions using chmod[3] and chown[4].

```
$ cd /data
$ sudo mkdir shared
$ sudo chown nobody:nogroup shared
$ sudo chmod g+ws shared
```

Here are 3 example directories to create for differing purposes:
'Music', 'Protected' and 'Test'
other examples might be 'Videos' and 'Pictures':

You will be able to create directories from your devices as well.

```
$ cd /data/shared
$ sudo mkdir Music Test
$ sudo chown nobody:nogroup Music Test
$ sudo chmod g+ws shared Music Protected

$ sudo mkdir Protected
$ sudo chown myname:mygroup Protected
$ sudo chmod g+ws Protected
```

I like having a general 'Protected' area that others can access but cannot change. I use secure-shell access to that area for dumping files that I manage without using Samba tools.

I use the Test area initially for testing from various devices; that is, create and delete files in the test directory.

As an example I put my old Music files in 'Music' so that it could be accessed from various devices 24x7. You can either keep the permissions as *nobody:nogroup*, allowing other people in your home network to help manage the collection, or you can change ownership so that only you manage them locally. In this example my login name is 'myname' with group 'mygroup':

```
$ du -sh /data/shared/Music/
7.4G    /data/shared/Music/
$ ls -la /data/shared/Music/
drwxr-sr-x  9 myname mygroup   4096 Apr 15 16:03 .
drwxrwsr-x  7 nobody nogroup   4096 Feb 26 11:59 ..
-rw-r--r--  1 myname mygroup 108364 Feb 27  2022 all.m3u
drwxr-xr-x  9 myname mygroup   4096 Feb 26  2022 Celtic
-rw-r--r--  1 myname mygroup  13373 Mar  6  2022 Celtic.m3u
...
drwxr-xr-x  5 myname mygroup   4096 Feb 27  2022 Nostalgia
-rw-r--r--  1 myname mygroup   6145 Feb 27  2022 Nostalgia.m3u
drwxr-xr-x 13 myname mygroup   4096 Feb 26  2022 Pop
-rw-r--r--  1 myname mygroup  10065 Feb 27  2022 Pop.m3u
drwxr-xr-x 39 myname mygroup   4096 Feb 26  2022 Rock
-rw-r--r--  1 myname mygroup  41656 Feb 27  2022 Rock.m3u
```

[3] changes the mode of a file or directory. It takes symbolic or numeric arguments.
[4] changes the owner of a file or directory; with a colon it also changes the group ownership.

Install the Samba software

Simply install the *samba* package; *apt* will pull in any dependencies:

```
$ sudo apt install samba
...
0 upgraded, 21 newly installed, 0 to remove and 3 not upgraded.
Need to get 7,870 kB of archives.
After this operation, 44.1 MB of additional disk space will be used.
Do you want to continue? [Y/n]
...
```

Modify the Samba configuration file

The main configuration file is:
/etc/samba/smb.conf

The file is organized into sections:

- the *global* section
- the *printers* section (which we will simply ignore, or you can comment it out)
- any other **shares** that you create; in this example I create one named *home*

Here are the specifics:

- Global Section
  1. In the global section change the *workgroup* name to something you like; I have chosen *LINUX*
  2. Just below the workgroup definition we add some vfs_fruit module options that allow Apple SMB clients to interact with the server
  3. we add a logging option to increase some logging for debugging purposes
- Our 'share' section named *home*

The modified smb.conf file is github.

```
$ cd /etc/samba
$ sudo cp -p smb.conf smb.conf.orig
$ sudo nano smb.conf
// The 'diff' command shows differences in snippets with the line numbers
// A more elegant way to see the differences would be side-by-side:
//   diff --color=always -y smb.conf smb.conf.orig | less -r
$ diff smb.conf smb.conf.orig
29,30c29
< #   workgroup = WORKGROUP
<    workgroup = LINUX
---
>    workgroup = WORKGROUP
34,37d32
<    fruit:nfs_aces = no
<    fruit:aapl = yes
<    vfs objects = catia fruit streams_xattr
<
68,69d62
<    log level = 1 passdb:3 auth:3
<
249,258d241
<
< [home]
<     comment = Samba on Raspberry Pi
<     path = /data/shared
<     writable = yes
<     read only = no
```

```
<     browsable = yes
<     guest ok = yes
<     create mask = 0664
<     directory mask = 0775
```

Start the Service and Run Some Tests

```
// enable the samba daemons
$ sudo systemctl enable smbd nmbd
$ sudo systemctl restart smbd nmbd
$ systemctl status smbd | grep Status:
     Status: "smbd: ready to serve connections..."

$ systemctl  status nmbd | grep Status:
     Status: "nmbd: ready to serve connections..."
```

Testing will depend on your device and client.

Suppose you have a MATE desktop session with your Pi server or another Linux device. Open a file browser:

Applications -> Accessories -> Files

The Files browser File menu has an option: *Connect to Server*. If you have an older version of Mate then find the help option and search for 'Connect to Server'.

A small connection window pops up. It is a bit annoying, so select any options that allow the file browser to remember your entries, and also create a bookmark.

There is no Samba password for 'guest', but the connection window will want one anyway; so give it the password 'guest'.

At this point an application named seahorse might pop up. It is the GNOME encryption interface, and you can store passwords and keys in it. I don't use it, but you might want to for this Samba share. You can always cancel the seahorse window. However it will reappear the next time you connect to the share.

For the connection request, fill in this data:

- Enter the IP address of your Pi server
- Select 'Type' Windows share
- Enter the share name: home
- Clear the Folder option
- Enter the domain name: LINUX (or whatever name you chose)
- User name: guest
- Password: guest (and select the option to remember it for seahorse)
- tick 'add bookmark' and give it a name

and finally connect.

Suppose you have an Android phone. Download the App: Cx File Explorer from your App Store. Under its *Network* tab you can open a 'remote' Samba share in your home network. You enter in the IP address and select 'Anonymous' instead of user/pass.

(!! get an example from windows and from an iphone)

Tests to run to validate functionality include the following:

1. Create a folder for your personal use
2. Browse to the Test folder
3. Copy and Paste a file from your device here
4. Create a folder here too, and copy your file into that folder
5. Delete all files and folders inside the Test folder

Backing up Your Server

Always, always, do some kind of backups on your server. For system backups, very little actually needs to be backed up, yet it is important to get into a frame of mind where you think about these things. Lets look at what you should back up on your server, and how you might do it.

There is no need to back up everything - you can always reinstall and reconfigure. This is my favourite list of system directories to back up:

- /etc – a lot of system configuration is in this directory. Some important configuration files found here are: the host's secure-shell keys, user account details, and most server configuration changes
- /home – this is where your user account resides
- /root – this is the superuser's home directory
- /var/log – system log files are here; for forensic reasons I back them up
- /var/spool – in case you have personalized cron job entries
- /var/www – if you have a web server then it's data files are usually here

If you are playing with database services then you need to inform yourself what directories and/or data exports should be used during backups. Note that when you have create a Samba or an NFS server you will have other data directories to back up, and these directories might be large.

An example backup script and process is in the appendix. We look at backing up both system and data directories.

Server Customization

Here is a list of tasks you can apply to your server for 24x7 service. Ubuntu installations are more common on laptops and desktops which are often connected via wireless, are turned on and off frequently, and have a lot of software configuration not usually present or needed on a server.

The main objective here is to show you some options that reduce complexity and memory consumption, and might improve security and reliability. You can always circle back here in the future and try them.

Remove anacron Service

UNIX and Linux has a mechanism called *cron* allowing servers to run commands at specific times and days. However personal and mobile computing is typically not powered on all the time. So operating systems like Linux have another mechanism called *anacron* which trys to run periodic cron-configured commands while the computer is still running. Since we are creating a 24x7 server we do not also need anacron – delete it:

```
$ sudo apt remove anacron
$ sudo apt purge anacron
```

Turn Off Bluetooth

If you won't be using it on your server then turn Bluetooth off.

The Pi does not have a BIOS like personal computers do; instead configuration changes to enable or disable devices are managed in the configuration file *config.txt* in */boot/firmware/*

You will need to reboot the server once you have make this change. If you also disable WiFi then wait until you have finished the next task.

```
// List bluetooth devices:
$ hcitool dev
Devices:
        hci0    E4:5F:01:A7:11:0F

// disable bluetooth services running on the Pi
$ sudo systemctl disable blueman-mechanism bluetooth

// Always save a copy of the original file with the 'cp' command:
$ cd /boot/firmware
```

```
$ sudo cp -p config.txt config.txt.orig
// Disable bluetooth in config.txt by adding 'dtoverlay=disable-bt' at the end
$ sudo nano config.txt
// Use the 'tail' command to see the end of the file:
$ tail -3 config.txt

dtoverlay=disable-bt
```

Turn Off Wireless

If you will use the built-in ethernet interface for networking on your server then turn WiFi off. I prefer wired connections for servers, especially since newer technology offers gigabit speed ethernet. In my experience, the network latency is usually better to wired devices. But if you prefer it on wireless then skip this task.

You will need to reboot the server once you have make this change, but **remember to connect the ethernet cable** on the Pi to your home router first.

```
// List wireless devices - after making this change you will not see this
// information:
$ iw dev
phy#0
        Unnamed/non-netdev interface
                wdev 0x2
                addr e6:5f:01:a7:71:0d
                type P2P-device
                txpower 31.00 dBm
        Interface wlan0
                ifindex 3
                wdev 0x1
                addr e4:5f:01:a7:71:0d
                ssid MYNET
                type managed
                channel 104 (5520 MHz), width: 80 MHz, center1: 5530 MHz
                txpower 31.00 dBm

// Disable wireless in config.txt by adding 'dtoverlay=disable-wifi' at the end
$ cd /boot/firmware
$ sudo nano config.txt
// Use the 'tail' command to see the end of the file:
# tail -3 config.txt

dtoverlay=disable-bt
dtoverlay=disable-wifi

// After rebooting the Pi disable the wireless connection and authentication
// service
$ sudo systemctl stop wpa_supplicant
$ sudo systemctl disable wpa_supplicant
```

Enable Boot-up Console Messages

Maybe like me you like seeing informational messages as a computer boots up. In that case you need to edit */boot/firmware/cmdline.txt* and remove the 'quiet' argument. On my system this one line file nows ends in:
'... fixrtc splash'
instead of
'... fixrtc quiet splash':

```
$ cd /boot/firmware
$ sudo cp -p cmdline.txt cmdline.txt.orig
$ sudo nano cmdline.txt
```

Disable the Graphical Login Interface

Simpler is better for a server. Normally 24x7 servers are headless, mouseless, keyboardless, and sit in the semi-darkness. A graphics-based console is therefore useless. Though your home server might not be as lonely as a data centre server, you might want to try a text-based console:

```
$ sudo systemctl set-default multi-user
$ sudo systemctl stop display-manager

// If you do have a mouse and a screen attached then you can still make
// the mouse work in a text console login -- it can be useful.  At work
// I have sometimes used the mouse at a console switch to quickly copy and
// paste process numbers for the kill command.

$ sudo apt install gpm
$ sudo systemctl enable gpm
```

Disabling Various Unused Services

Here are some services which normally can be disabled. Of course, if any of these services are interesting to you then keep them. Note that server processes are sometimes called *daemons*.

You can handle multiple services at a time, but doing them individually allows you to watch for any feedback. You can also simply disable these services without stopping them. They will not run on the next reboot.

```
// If you want to run a series of commands as root you can sudo to the bash
// shell, run your commands, and then exit the shell.  Be careful to
// always exit immediately after running your commands.
$ sudo /bin/bash

// disable serial and bluetooth modems or serial devices
# systemctl stop ModemManager
# systemctl disable ModemManager
# systemctl stop hciuart
# systemctl disable hciuart

// disable VPN and printing services - you can print without running
// a local printer daemon (!!maybe document using one tho )
# systemctl stop openvpn
# systemctl disable openvpn
# systemctl stop cups-browsed cups
# systemctl disable cups-browsed cups

// disable secureboot-db
// UEFI Secure boot
// (!! add notes)

// disable whoopsie and kerneloops if you don't want to be sending
// information to outside entities
# systemctl stop kerneloops
# systemctl disable kerneloops
# systemctl stop whoopsie
# systemctl disable whoopsie
```

```
# apt remove whoopsie kerneloops
# apt purge whoopsie kerneloops
```

Enabling the Secure Shell Daemon

describe enabling, configuring and creating a key pair.

Enabling Remote Desktop Services

description here for startx, x2go and xrdp.

Creating a Web Site on the Server

description here.

Enabling a Git Service and Browsing It on Your Web Site

description here.

Starting Up an NFS Service for Other Linux Devices

description here.

Other Possible Services

description here - eg, MySQL, PHP, …

Appendix

Identifying Device Names for Storage Devices

In a Linux system it is important to correctly identify storage devices, especially when you want to repartition or reformat them. If you pick the wrong device you might wipe out its data.

Start with *lsblk* to list all current block (storage) devices. Note that a microSD card in a microSD slot will be identified with a name starting with */dev/mmcblk*. But if you insert the microSD card into a multi-slot card reader then the Linux kernel will identify any kind of inserted card in the reader as a generic 'scsi disk' type and it will appear with a name which starts with */dev/sd*.

Here is an example of a Pi that has 3 storage disks and a USB card reader with 4 slots. The lsblk command also shows active mountpoints. The 3 disks are:

- the Pi's system microSD card (mmcblk0) with 2 partitions mounted as / and */boot/firmware*
- a USB stick (sda) with 1 partition mounted as */usbdata*
- a card reader with 4 slots – 3 slots have 0 bytes, so they are empty (sdb, sdc, sdd) and one of the slots (sde) is occupied by another 256 GB microSD card which is listed with lesser size of 232 GB.
- you can use *fdisk* and *parted* to look more closely at the sde device:

```
# lsblk -i
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda           8:0    1 238.5G  0 disk
`-sda1        8:1    1 238.5G  0 part /usbdata
sdb           8:16   1     0B  0 disk
sdc           8:32   1     0B  0 disk
sdd           8:48   1     0B  0 disk
sde           8:64   1 231.7G  0 disk
|-sde1        8:65   1   243M  0 part
`-sde2        8:66   1   5.9G  0 part
mmcblk0     179:0    0  59.4G  0 disk
```

```
|-mmcblk0p1 179:1    0   243M  0 part /boot/firmware
`-mmcblk0p2 179:2    0  59.2G  0 part /

# fdisk -l /dev/sde
Disk /dev/sde: 231.68 GiB, 248765218816 bytes, 485869568 sectors
Disk model: FCR-HS3       -3
...
Disklabel type: dos
Disk identifier: 0x11d94b9e

Device     Boot  Start      End  Sectors  Size Id Type
/dev/sde1  *      2048   499711   497664  243M  c W95 FAT32 (LBA)
/dev/sde2       499712 12969983 12470272  5.9G 83 Linux

# parted /dev/sde print
Model:  FCR-HS3 -3 (scsi)
Disk /dev/sde: 249GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
 1      1049kB  256MB   255MB   primary  fat16        boot, lba
 2      256MB   6641MB  6385MB  primary  ext4
```

You can pass options to the *lsblk* command so that you print out columns you are interested in – try *lsblk --help* to see other options.

```
# lsblk -i -o 'NAME,MODEL,VENDOR,SIZE,MOUNTPOINT,FSTYPE'
NAME        MODEL       VENDOR     SIZE MOUNTPOINT     FSTYPE
sda         Extreme Pro SanDisk  238.5G
`-sda1                           238.5G /usbdata       ext4
sdb         FCR-HS3 -0               0B
sdc         FCR-HS3 -1               0B
sdd         FCR-HS3 -2               0B
sde         FCR-HS3 -3           231.7G
|-sde1                             243M                vfat
`-sde2                             5.9G                ext4
mmcblk0                           59.4G
|-mmcblk0p1                        243M /boot/firmware vfat
`-mmcblk0p2                       59.2G /              ext4
```

Installation Disk Creation from the Command-line

```
// create a directory for downloaded images
$ mkdir raspberry-pi-images
$ cd raspberry-pi-images

// Use wget to pull the compressed image into our directory
// (The web page also shows the file checksum so that you can verify
//  that the compressed file is not corrupted)
$ serverpath="https://releases.ubuntu-mate.org/jammy/arm64"
$ compressed="ubuntu-mate-22.04-desktop-arm64+raspi.img.xz"
$ wget -N -nd $serverpath"/"$compressed
$ sha256sum ubuntu-mate-22.04-desktop-arm64+raspi.img.xz
3b538f8462cdd957acfbab57f5d949faa607c50c3fb8e6e9d1ad13d5cd6c0c02 ...
```

```
// uncompress the file so we can write the image file to our microSD.
// the 1.9 GB compressed file uncompressed to 6.2 GB:
$ xz -dv ubuntu-mate-22.04-desktop-arm64+raspi.img.xz
ubuntu-mate-22.04-desktop-arm64+raspi.img.xz (1/1)
  5.1 %      95.2 MiB / 404.4 MiB = 0.235    35 MiB/s     0:11   3 min 40 s
  ...
  100 %   1,847.8 MiB / 6,333.0 MiB = 0.292  27 MiB/s     3:50
```

```
Now plug in the microSD card,
[identify the microSD card device name](#find-device), and then
use the *dd* command to write the image to it.  Safely remove the
device when you are done with the *eject* command.  In this example
the device name */dev/sdX* is a place-holder for the real device name:
```

```
$ img="ubuntu-mate-22.04-desktop-arm64+raspi.img"
$ sudo dd if=$img of=/dev/sdX bs=32M conv=fsync status=progress
$ sudo eject /dev/sdX
```

Modify the Partitioning of the Installation Image

If you modify the microSD's partioning *before* you start the installation then you can reserve a portion of the disk for special data usage - for example as a shared Samba area or an NFS area. We do this by expanding the main Linux partition up to 40 GB, and then we create a third partition.

I show here how to do that from another Linux computer (in my case another Pi) with a USB card reader and an inserted 256 GB microSD card.

There is a good graphical tool named *gparted* which is easy to use. Beginners should certainly use it, and it is great when managing a handful of servers (it is a different story if you are managing dozens or thousands of servers).

A few gparted notes: You will need to first install it with *sudo apt install gparted*
    If you opt for this tool then it is all you need
    It is intuitive, and there are many tutorials on the web
    Be careful to pick the correct disk from the drop-down list
    Expand the second partition, then create the third (data) partition
    Also use the tool to create an *ext4* file system on the third partition once you have created it

As always, I show the command-line example in this section. As a bonus it shows a common problem of dealing with partition alignment when manually editing partitions. Skip the rest of this section if you have opted for gparted.

Here are some common command-line tools to help us:

lsblk this command will list all block devices
fdisk this interactive text-based command can change disk partitioning
    To show all disk and their partitions, do: fdisk -l
    You can only operate on one disk
parted this interactive text-based command can also change disk partitioning
    You can only operate on one disk
mkfs.ext4 You should create an ext4 file system on the new partition
Utilities used to find disk information: lshw -C disk
    hdparm -I /dev/sdX (where X is a block device letter)
    smartctl -a /dev/sdX (needs *smartmontools* to be installed)

Identify the Main Linux Partition on the microSD First we need to identify the device name, and then we use that device name in the partitioning tool. In my test situation I am using */dev/sde* and I am targeting the main Linux (second) partition: */dev/sde2*.

Expand the Main Linux Partition on the microSD 40 GB is lots of space for future system needs, so I decide to expand the second partition from 6 up to 40 GB.

To be sure this partition is sound I run a check on it with *e2fsck*. Then I use *parted* to expand the partition, and then *resize2fs* which can adjusts the size of the underlying ext4 filesystem.

```
// run a filesystem check on the target partition:
$ sudo e2fsck /dev/sde2
e2fsck 1.46.5 (30-Dec-2021)
writable: clean, 224088/390144 files, 1485804/1558784 blocks

// invoke the partitioning tool *parted*, print the partition table
// for reference, and then resize the second partition:
$ sudo parted /dev/sde
...
(parted) print
...
Number  Start   End     Size    Type     File system  Flags
 1      1049kB  256MB   255MB   primary  fat16        boot, lba
 2      256MB   6641MB  6385MB  primary  ext4

(parted) resizepart
Partition number? 2
End?  [6641MB]? 40G

(parted) print
...
Number  Start   End     Size    Type     File system  Flags
 1      1049kB  256MB   255MB   primary  fat16        boot, lba
 2      256MB   40.0GB  39.7GB  primary  ext4

(parted) quit

// Now expand the underlying filesystem to the end of the partition
$ sudo resize2fs /dev/sde2
resize2fs 1.46.5 (30-Dec-2021)
Resizing the filesystem on /dev/sde2 to 9703161 (4k) blocks.
The filesystem on /dev/sde2 is now 9703161 (4k) blocks long.
```

Create a New Data Partition Now we want to create a third large partition. We run into the problem of partition alignment because I chose 40 GB for the second partition expansion without considering alignment for the next partition.

```
// invoke *parted* and print the partition table in sector units:
$ sudo parted /dev/sde
...
(parted) unit s print
Model:  FCR-HS3 -3 (scsi)
Disk /dev/sde: 485869568s
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start    End        Size       Type     File system  Flags
 1      2048s    499711s    497664s    primary  fat16        boot, lba
 2      499712s  78125000s  77625289s  primary  ext4

// the 'End' of the second partition is at 78125000 sectors, so I increment
// that number and try to create the third partition up to 100% of the disk
// There is a warning about partition alignment, so I cancel that operation
```

```
(parted) mkpart primary ext4 78125001 100%
Warning: The resulting partition is not properly aligned for best performance:
78125001s % 2048s != 0s
Ignore/Cancel? c
```

Disk technology has evolved considerably. Block devices traditionally had a default 512 byte sector size (a sector is the minimum usable unit), but newer disks may have a 4096 (4k) sector size. In order to work efficiently with different sector sizes on various disk technologies a good starting point is at sector 2048. This is at 1 mebibyte (MiB), or 1048576 bytes into the disk, since 512 bytes * 2048 sectors is 1048576 bytes. You lose a bit of disk space at the 'front' of the disk, but partitioning and file system data structures are better aligned, and disk performance is enhanced.

```
// To calculate the best starting sector number for an aligned new
//  partition simply calculate:
//  TRUNCATE(FIRST_POSSIBLE_SECTOR / 2048) * 2048 + 2048
//  thus: 78125001 / 2048 = 38146.973144
//        TRUNCATE(38146.973144) = 38146
//        (38146 * 2048) + 2048  = 78125056

(parted) mkpart primary ext4 78125056 100%
(parted) print unit s
...
Number  Start   End     Size    Type     File system  Flags
 1      1049kB  256MB   255MB   primary  fat16        boot, lba
 2      256MB   40.0GB  39.7GB  primary  ext4
 3      40.0GB  249GB   209GB   primary
(parted) align-check optimal 3
3 aligned

(parted) unit s print free
...
Number  Start      End         Size        Type     File system  Flags
        32s        2047s       2016s                Free Space
 1      2048s      499711s     497664s     primary  fat16        boot, lba
 2      499712s    78125000s   77625289s   primary  ext4
        78125001s  78125055s   55s                  Free Space
 3      78125056s  485869567s  407744512s  primary

(parted) quit
```

Though we have now a bit of wasted space between partition 2 and 3, we can extend partition 2 to use that space. We need to resize its ext4 file system after:

```
// first check the file system:
$ sudo e2fsck /dev/sde2
e2fsck 1.46.5 (30-Dec-2021)
writable: clean, 224088/2414016 files, 1615846/9703161 blocks

$ sudo parted /dev/sde
...
(parted) unit s print free
...
Number  Start      End         Size        Type     File system  Flags
        32s        2047s       2016s                Free Space
 1      2048s      499711s     497664s     primary  fat16        boot, lba
 2      499712s    78125000s   77625289s   primary  ext4
        78125001s  78125055s   55s                  Free Space
```

```
 3      78125056s  485869567s  407744512s  primary

(parted) resizepart
Partition number? 2
End?  [78125000s]? 78125055s
(parted) print free
...
Number  Start      End         Size        Type     File system  Flags
        32s        2047s       2016s                Free Space
 1      2048s      499711s     497664s     primary  fat16        boot, lba
 2      499712s    78125055s   77625344s   primary  ext4
 3      78125056s  485869567s  407744512s  primary

(parted) quit
Information: You may need to update /etc/fstab.

$ sudo resize2fs /dev/sde2
resize2fs 1.46.5 (30-Dec-2021)
Resizing the filesystem on /dev/sde2 to 9703168 (4k) blocks.
The filesystem on /dev/sde2 is now 9703168 (4k) blocks long.
```

Create a Filesystem on the New Data Partition Ubuntu uses the *ext4* filesystem, so let's create that filesystem on the new data partition:

```
$ sudo mkfs.ext4 /dev/sde3
mke2fs 1.46.5 (30-Dec-2021)
Creating filesystem with 50968064 4k blocks and 12746752 inodes
Filesystem UUID: 2dba1eef-34e4-47ed-90fc-c1938d5fa9e0
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
    4096000, 7962624, 11239424, 20480000, 23887872

Allocating group tables: done
Writing inode tables: done
Creating journal (262144 blocks): done
Writing superblocks and filesystem accounting information: done
```

By default, the generations of the *ext* filesystem reserve 5% of the available space for the *root* user. On a data partition where most files created will belong to ordinary users this reservation is not necessary. 5% of 200 GB is 10 GB - that is a lot of space. So it is a good idea to reduce the percentage to 1%; do this with the *tune2fs* utility:

```
$ sudo tune2fs -l /dev/sde3 | grep -i count
...
Reserved block count:     2548403

$ sudo tune2fs -m 1 /dev/sde3
tune2fs 1.46.5 (30-Dec-2021)
Setting reserved blocks percentage to 1% (509680 blocks)

$ sudo tune2fs -l /dev/sde3 | grep -i count
...
Reserved block count:     509680
```

Lastly, lets add a *label* to this partition to make it easier to mount the filesystem in the future. We will use the label *PI-DATA*:

```
$ sudo tune2fs -L PI-DATA /dev/sde3
tune2fs 1.46.5 (30-Dec-2021)
```

Setting Up a Data Area

If you did not modify the initial partitioning of your microSD card then you will simply make a directory in the root of your filesystem where we will store any data associated with a Samba service or with an NFS service. In case you did make a data partition on the microSD card then we *still* need to make a directory in the root of the filesystem to mount that data partition:

Let's call the directory */data*:

```
$ sudo mkdir /data
```

For the case where you created the third (data) partition then we need to mount it and make the mount action permanent on reboots. For this we create an entry in the filesystem table, the */etc/fstab* file:

```
// make a copy first
$ sudo cp -p /etc/fstab /etc/fstab.orig

// There are 6 fields in an fstab entry:
//    1. the partition name, the partition label or the partition uuid
//    2. the directory to use for the mount point
//    3. the filesystem type
//    4. any mount options recognized by the mount command
//    5. use a zero here, it is a legacy option for the 'dump' command
//    6. the file system check ordering, use '2' here

// Edit the file, adding a mount entry line to the end of the file;
// recall that we added the label 'PI-DATA' to its filesystem:
$ sudo nano /etc/fstab
$ tail -2 /etc/fstab
LABEL=PI-DATA       /data           ext4    defaults  0  2

$ sudo mount /data
$ ls -la /data
total 24
drwxr-xr-x  3 root root  4096 Apr 16 10:35 .
drwxr-xr-x 20 root root  4096 Apr 16 14:30 ..
drwx------  2 root root 16384 Apr 16 10:35 lost+found

$ df -h /data
Filesystem      Size  Used Avail Use% Mounted on
/dev/sde3       191G   28K  189G   1% /data (!! put correct dev here)
```

Some Command-line Utilities and Their Purpose

Command	Purpose
whoami	Shows your login name
id	Shows your UID and GID numbers and all of your group memberships
pwd	Shows your current working directory
ls	Shows a listing of your current directory
ls -l	Shows a detailed listing of your current directory
ls /	Shows a listing of the base of the file system
ls /root	Try to show a listing of the superuser's home directory
sudo ls /root	Enter your password to show that listing
man sudo	Shows the manual page for the sudo command (type q to quit)
date	Shows the current date and time
cat /etc/lsb-release	Shows the contents of this file
uptime	Shows how long this computer has been up
cd /tmp	Change directories to the 'tmp' directory

Command	Purpose
touch example	Creates a new (and empty) file named ‘example’
rm -i example	Removes the file named ‘example’ if you respond with: y
mkdir thisdir	Creates a new directory named ‘thisdir’
mkdir --help	Shows help on using the mkdir command
rmdir thisdir	Removes the directory named ‘thisdir’
cd	Without an argument, it takes you back to your home
file .bash*	Shows what kind of files whose names start with ‘.bash’
echo $SHELL	Shows what shell you use
env | sort | less	Shows your environment variables, sorted (q to quit)

A MATE Configuration Exercise

Here are a series of exercises you can try on a fresh installation of the MATE desktop:

I always delete the bottom panel and reconfigure the top panel (right-click on the top panel and add an applet). I prefer the ‘Classic Menu’ setup which splits menus into applications, places and system options. You can change the look and feel; you can remove the password request on the screensaver configuration; you can change the background, etc. Search online for videos showing configuration sessions. Also add some custom panel buttons.

An Example Process and Script for Backups

System Backups There are many ways to do backups - this is just one example.

First make the top directory with sufficient space to hold one week’s worth of backups. The files will be compressed so it requires only a few hundred megabytes per day for each day of the week:

```
# mkdir /var/local-backups
```

(this will become a link instead into github)