Creating a Linux Home Server

Denice Deatrich

Created: March 2023 Last update: June 14, 2023

Table of 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 still running Ubuntu LTS 16.04, and I will document it as well as I update it. 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. There is also a lot of detail in this guide; this is my personal preference. I grow tired of the current trend to provide internet-searched answers in a few phrases to fit on the screen of a mobile phone.

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 (**...**)

Sometimes I add a double-exclamation (**!!**) mark somewhere – this is only a reminder for myself to fix an issue at that point in the documentation.

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

Picking the OS (Operating System) and the Window Manager

Though many Raspberry Pi owners run the Raspberry Pi OS (formerly known as Raspbian), 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.

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, as indicated by the release number.

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. MATE is not flashy, but it gets the job done.

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.

This installation image still uses the X.org display server instead of Wayland, partly because it uses MATE which is not yet ready for Wayland at this Ubuntu LTS release, and also because this build is for the Raspberry Pi, where Wayland support is new. Moreover remote desktop support is a work in progress for Wayland environments, and is better left to the X.org protocol for now.

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

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 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 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 with a password – 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.

Before going further immediately update the software on the system. The MATE installation image is not released often, so it can be a bit behind the package update curve. As well, any final configuration issues will be updated.

Open a terminal window by selecting:
Application -> System Tools -> MATE Terminal
and enter the following commands:

```
// This will update the system's knowledge about what should be updated;
// in other works, the cache of software package names pending for update:
$ sudo apt update

// then update the software; the command is actually 'upgrade', which is odd,
// at least to me..
$ sudo apt upgrade
```

It will take a while. Once finished reboot the system to get the newer kernel. On the far upper right taskbar, select the powerbutton icon, and then select *Switch Off* -> *Restart*.

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 well 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 to find out how to create your 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[2] and chown[3].

```
$ 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 inside the shared area using your other devices as well.

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

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

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.

```
$ cd /data/shared
$ sudo mkdir Protected
$ sudo chown myname:mygroup Protected
$ sudo chmod g+ws Protected
```

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

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
```

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 in 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.orig smb.conf | less -r
$ diff smb.conf.orig smb.conf
29c29,30
<    workgroup = WORKGROUP
---
> #   workgroup = WORKGROUP
>    workgroup = LINUX
33a35,39
> # for Apple SMB clients
>    fruit:nfs_aces = no
>    fruit:aapl = yes
>    vfs objects = catia fruit streams_xattr
>
62a69,70
>    log level = 1 passdb:3 auth:3
>
241a250,259
>
> [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 on your Pi server or on 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' to make it happy.

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.

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 in smb.conf)
- 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. Let's 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 which directories and/or data exports should be added and/or used for 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.

A backup process and a link to an example backup script is in the appendix. We look at backing up both system and data directories, including the backup of large directories. Compressed system directories are typically small. But /home and /data/shared might be large. Be aware of your space needs, and adjust backups accordingly.

Recovering Files from Backups

Recovering files from backups is fairly easy. It is best to use an empty directory with adequate space. In this example we recover files from a compressed tar file. We unpack the tarball in the empty directory, and then move or copy any files into place in the file system. Example:

```
// This example unpacks home.tgz and etc.tgz
$ sudo mkdir /var/local-recovery

// We can copy files from the local backup tree: /var/local-backups/
// or from the external drive once we mount it:  /mnt/backups/
// So use the needed pathname instead of '/path/to/' below:
$ sudo cp -ip /path/to/home.tgz /path/to/etc.tgz  /var/local-recovery/

$ cd /var/local-recovery
$ sudo tar -zxf etc.tgz
$ sudo tar -zxf home.tgz
$ sudo rm etc.tgz home.tgz
$ ls -l
drwxr-xr-x 152 root root 12288 Jun  1 10:56 etc
drwxr-xr-x   3 root root  4096 May 25 10:45 home
$ ls -l home/
drwxr-x--- 25 myname myname 4096 Jun  1 21:12 myname
```

If you need to recover files from large directory backups then you can copy the files directly from the removable media to your target directories.

```
// This example recovers a directory inside the large directory backup of
// /home.  First mount the USB drive -- the example partition here is at
// /dev/sda1:
$ sudo mount /dev/sda1 /mnt
$ cd /mnt/rsyncs
$ ls
0  3  5  copy
$ cd 0/home/myname/
$ ls
bin  doc  downloads  etc  git  icons  inc  lib  src

// Since the files are my files then I do not need to use 'sudo'.
// Note that I copy it to a different directory name so that I have the
// option of comparing any existing 'bin' directory in my home.
$ cp -a bin ~/bin.recovered

// Change directories to away from the USB drive so that you can unmount it:
$ cd
$ sudo umount /mnt
$ pwd
/home/myname
$ diff -r bin bin.recovered
```

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.

By all means, ignore all of this if you don't want to be bothered with disabling extraneous software. I have spent 3 decades managing UNIX and Linux systems, so I can be a bit picky about what runs on my systems.

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 eventually reboot the server once you have make this change. If you also disable WiFi then wait until you have finished the next task, or any other tasks in this chapter.

```
// 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 to keep the server 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 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
```

Disable Snap Infrastructure

Ubuntu promotes another kind of software packaging called **Snaps** (which includes an *App* store). Some users are not pleased with issues introduced by the underlying support software, and decide to delete Snap support from their systems. In my early testing of Ubuntu LTS 22.04 I also ran into the problem of firefox not able to start, and like others I traced it back to snap (firefox is installed from a Snap package).

I don't think 'Snaps' are meant for a server environment, and so I remove the associated software. I certainly find it distasteful to have more than a dozen mounted loop devices cluttering up output of block device commands for just a handful of snap packages. I would rather free up the memory footprint and inodes for other purposes.

So here is a quick summary on removing Snap support, that is, all snap packages and the snapd daemon:

Disable the daemon

You should close firefox if it is running in a desktop setting. Then disable the snapd services and socket:

```
$ sudo systemctl disable snapd snapd.seeded snapd.socket
```

Remove the packages

List the Snap packages installed, and then delete them. Leave *base* and *snapd* packages until the end. As noted in [Erica's instructions][snap-remove] remove packages one at a time and watch for messages warning about dependencies. This is my list of Snap packages; your list might be different depending on what you have installed:

```
$ snap list
Name                       Version            ...  Publisher      Notes
bare                       1.0                ...  canonical     base
core20                     20230404           ...  canonical     base
core22                     20230404           ...  canonical     base
firefox                    112.0.2-1          ...  mozilla       -
gnome-3-38-2004            0+git.6f39565      ...  canonical     -
gnome-42-2204              0+git.587e965      ...  canonical     -
gtk-common-themes          0.1-81-g442e511    ...  canonical     -
snapd                      2.59.2             ...  canonical     snapd
snapd-desktop-integration  0.9                ...  canonical     -
software-boutique          0+git.0fdcecc      ...  flexiondotorg classic
ubuntu-mate-pi             0+git.0f0bcdf      ...  ubuntu-mate   -
ubuntu-mate-welcome        22.04.0-a59036a6   ...  flexiondotorg classic

$ sudo snap remove firefox
$ sudo snap remove software-boutique
$ sudo snap remove ubuntu-mate-welcome
$ sudo snap remove ubuntu-mate-pi
$ sudo snap remove snapd-desktop-integration
$ sudo snap remove gtk-common-themes
$ sudo snap remove gnome-42-2204
$ sudo snap remove gnome-3-38-2004
$ sudo snap remove core22
$ sudo snap remove core20
```

```
$ sudo snap remove bare
$ sudo snap remove snapd

$ snap list
No snaps are installed yet. Try 'snap install hello-world'.
```

Clean up and add a new firefox dpkg source

Completely remove snapd and its cache files from the system. Then add configuration files for *apt* access to firefox *dpkg-based* packages. Finally install firefox from the Mozilla Personal Package Archive (PPA):

```
$ sudo apt autoremove --purge snapd
$ sudo rm -rf /root/snap
$ rm -rf ~/snap

// Create the necessary apt configurations for firefox:

$ sudo nano /etc/apt/preferences.d/firefox-no-snap
$ cat /etc/apt/preferences.d/firefox-no-snap
Package: firefox*
Pin: release o=Ubuntu*
Pin-Priority: -1

$ sudo add-apt-repository ppa:mozillateam/ppa
...
PPA publishes dbgsym, you may need to include 'main/debug' component
Repository: 'deb https://ppa.launchpadcontent.net/.../ppa/ubuntu/ jammy main'
Description:
Mozilla Team's Firefox stable + 102 ESR and Thunderbird 102 stable builds
Support for Ubuntu 16.04 ESM is included.
...

// Install firefox
$ sudo apt install firefox
```

Modify the Swap Setup

There is a big 1 GB swapfile in the root of the filesystem - I find that offensive, so I moved it. If you are not as easily offended as I am then skip this topic.

It is a good idea to have some kind of swap enabled, since swap is only used if too much memory is being consumed by processes. Once memory is low the system will start using any configured swap on disk. Of course this is slower than memory, but it is better to use some swap at those moments instead of having an unfortunate process die because of an out-of-memory condition.

Over time if you never see swap being used then you could turn swap off and delete the swap file.

Here we create another 1 GB file – it can be much larger if needed; but if you need a lot of swap then you should investigate to see what is eating memory.

```
// check to see what the current swap usage is; in this case it is 0
$ free -t
           total        used        free      shared  buff/cache   available
Mem:     3881060      176900     3022600        5332      681560     3541748
Swap:    1048572           0     1048572
Total:   4929632      176900     4071172

// Look at what systemd does with swap, and turn off the appropriate items
```

```
$ systemctl list-unit-files | grep swap
mkswap.service                              disabled        enabled
swapfile.swap                               static          -
swap.target                                 static          -

// There will be a .swap rule for every swap file - this one is for /swapfile
// and we want to get rid of it
$ sudo systemctl mask swapfile.swap
Created symlink /etc/systemd/system/swapfile.swap → /dev/null.

// Even though the 'mkswap' service is by default disabled, I also
// mask it so that it doesn't come back from the dead - because it
// will come back if you don't also mask that service
$ sudo systemctl mask mkswap.service
Created symlink /etc/systemd/system/mkswap.service → /dev/null.

// turn current swap off so we can delete the old file
$ sudo swapoff -a
$ sudo rm /swapfile

// Create an new swapfile in a subdirectory
$ sudo mkdir /swap
$ sudo fallocate -l 1G /swap/swapfile
$ sudo mkswap /swap/swapfile
Setting up swapspace version 1, size = 1024 MiB (1073737728 bytes)
no label, UUID=3e64d157-6f09-48d1-94c2-3851b82a73b7

// Protect the swap file and add it to /etc/fstab
$ sudo chmod 600 /swap/swapfile
$ sudo nano /etc/fstab
$ grep swap /etc/fstab
/swap/swapfile    none           swap defaults     0 0

// Turn swap back on and check
$ sudo swapon -a
$ swapon
NAME           TYPE  SIZE USED PRIO
/swap/swapfile file 1024M   0B   -2

// Note that systemd will show a new 'swap' type named 'swap-swapfile.swap'
$ systemctl --type swap
  UNIT               LOAD   ACTIVE SUB    DESCRIPTION
  swap-swapfile.swap loaded active active /swap/swapfile
```

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
```

Disable 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*.

The *systemctl* command can handle multiple services at the same 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 System Security Services Daemon ([sssd][sssd]) if you don't need it
# systemctl disable sssd

// Disable UEFI Secure Boot (secureboot-db)
//
# systemctl disable secureboot-db

// 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

// If you want to disable other apport-based crash reporting then remove apport
// from your server:
# apt autoremove --purge apport
```

Miscellaneous Configuration Tweaks

Change Local Time Presentation Globally

If you prefer to see time in 24 hour format, or if you prefer to tweak other locale settings, then use *localectl* to set global locale settings.

In this example the locale setting is generic English with a region code for Canada. Because the British English locale uses a 24 hour clock then changing only the time locale will show datestrings with a 24 hour clock:

```
// Show your current locale settings
$ locale
LANG=en_CA.UTF-8
LANGUAGE=en_CA:en
LC_CTYPE="en_CA.UTF-8"
LC_NUMERIC="en_CA.UTF-8"
LC_TIME=en_CA.UTF-8
LC_TIME=en_GB.UTF-8
...

// What the date is in the current (Canadian) locale
$ date
Thu 11 May 2023 08:43:57 AM MDT

// What the date would look like if (American) en_US.UTF-8 were used:
$ LC_TIME=en_US.UTF-8 date
Thu May 11 08:45:18 AM MDT 2023

// What the date would look like if (British) en_GB.UTF-8 were used:
$ LC_TIME=en_GB.UTF-8 date
Thu 11 May 08:44:00 MDT 2023

// Change it to the British style.  The change is immediate, but since
// you inherit the older locale environment at login, then you will not see
// the change until you logout, and then back in.
$ sudo localectl set-locale LC_TIME="en_GB.UTF-8"
```

Change Log Rotation File Naming

Ubuntu installs a log rotation package which controls how log files are rotated on your server. This package typically once a week compresses log files to a different name in /var/log/ and truncates the current log. The resulting files are rotated through a specified rotation, and the oldest compressed log is deleted; for example here are 4 weeks worth of rotated auth.log files:

```
$ ls -ltr /var/log/ | grep auth.log
-rw-r-----  1 syslog     adm      3498 Apr 15 23:17 auth.log.4.gz
-rw-r-----  1 syslog     adm      8178 Apr 22 23:17 auth.log.3.gz
-rw-r-----  1 syslog     adm      7225 Apr 30 01:09 auth.log.2.gz
-rw-r-----  1 syslog     adm     58810 May  7 00:22 auth.log.1
-rw-r-----  1 syslog     adm     46426 May 11 09:17 auth.log
```

A better scheme is to use the ‘dateext’ option in */etc/logrotate.conf* so that older compressed logs keep their compressed and dated names until they are deleted:

```
$ ls -ltr /var/log/ | grep auth.log
-rw-r----- 1 syslog      adm      3287 Apr 17 07:30 auth.log-20230417.gz
-rw-r----- 1 syslog      adm      2494 Apr 23 07:30 auth.log-20230423.gz
-rw-r----- 1 syslog      adm      4495 May  1 07:30 auth.log-20230501.gz
-rw-r----- 1 syslog      adm     35715 May  7 07:30 auth.log-20230507
-rw-r----- 1 syslog      adm     29500 May 11 09:55 auth.log
```

To make this change /etc/logrotate.conf is modified. We also set the number of rotations to keep to 12 weeks instead of 4 weeks. Note that per-service log file customization is possible; look at examples in */etc/logrotate.d/*

```
$ cd /etc
$ sudo cp -p logrotate.conf logrotate.conf.orig
$ sudo nano logrotate.conf
$ diff logrotate.conf.orig logrotate.conf
```

```
13c13,14
< rotate 4
---
> #rotate 4
> rotate 12
19c20
< #dateext
---
> dateext
```

Other Configuration Issues

These topics will be documented soon:

- automatically or manually managing software updates
- explore firewall issues - ufw seems lacking
- getting rid of ESM messages in terminal logins
- local time configuration and ntp configuration

Enabling the Secure Shell Daemon and Using Secure Shell

The Secure Shell daemon, ***sshd***, is a very useful and important service for connecting between computers near or far. If you are never going to connect via SSH into your Pi home server from within your network then do NOT install the daemon. You can always use the secure shell client, ***ssh***, to initiate a connection to some external server – for that you do not need sshd.

Configure the sshd Daemon

If you will be needing sshd then first install it, since it is not installed by default in the LTS desktop version:

```
$ sudo apt install openssh-server
```

If you will be using ssh to connect to any local Linux systems, then think about configuring your local area network (LAN) to suit your taste. There is an explanation in the appendix.

There are some sshd configuration issues that I like to fix in the secure shell daemon's configuration file: */etc/ssh/sshd_config*. The issues to fix are:

- stop the daemon from listening on IPv6: *AddressFamily inet*
- tell the daemon to use DNS: *UseDNS yes*
- limit ssh access to yourself in your LAN; and block ssh access to the root user except for 'localhost': AllowUsers myname@192.168.1.* *@localhost*

```
$ cd /etc/ssh
$ sudo cp -p sshd_config sshd_config.orig

// edit the file:
$ sudo nano sshd_config

$ diff sshd_config.orig sshd_config
15a16
> AddressFamily inet
101a103
> UseDNS yes
122a125,130
>
> # Limit access to root user; only local users can connect via ssh
> # to root only if root's authorized_keys file allows them.
> # note: using @localhost does not work on ubuntu unless you set UseDNS to yes
> AllowUsers    myname@192.168.1.* *@localhost
>

// Another option is to also allow root access to this server from your Linux
```

```
// desktop (eg: 192.168.1.65).  Then the 'AllowUsers' configuration would
// look like this:
AllowUsers      myname@192.168.1.* root@192.168.1.65 *@localhost
```

Configure a Personal SSH Key Pair

If you use the Linux command-line for work between computers you soon understand the usefulness of ssh. Here we go through the exercise of creating an ssh key pair so that you can connect securely between devices.

We use *ssh-keygen* to create the key pair. You should treat the private key carefully, distributing it to your desktop systems only. You can copy your public key to remote hosts, where you create an *authorized_keys* file that specifies which public keys are allowed to connect without using a standard system password.

Over time the Secure Shell key types have changed. Some key types are no longer considered secure, like SSH-1 or DSA keys. Here we will use an SSH RSA-based key with a key size of 4096 bits. Always use a strong passphrase. It is not like a password since you have more liberty to use combinations of characters; as the man-page on ssh-keygen says:

> A passphrase is similar to a password, except it can be a phrase with a series of words, punctuation, numbers, whitespace, or any string of characters you want.

I would not create a passphrase that is less than 16 characters; I would certainly never set an empty passphrase.

```
// If you do not yet have a .ssh directory in your home directory then
// create one now; and give access to yourself only:

$ cd
$ mkdir .ssh
$ chmod 700 .ssh

// Generate the key:
$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/myname/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/myname/.ssh/id_rsa.

// The private key is named 'id_rsa' and the public key is named 'id_rsa.pub'
// Note the permissions of these 2 files; the private key is protected
// and is read-write only to the owner.
$ ls -l ~/.ssh/id_rsa ~/.ssh/id_rsa.pub
-rw------- 1 myname myname 3326 May  2 22:34 /home/myname/.ssh/id_rsa
-rw-r--r-- 1 myname myname  746 May  2 22:34 /home/myname/.ssh/id_rsa.pub
```

Be sure to back-up important directories like $HOME/.ssh – in your home environment it might not seem important, but once you start using your ssh keys for access to external resources then you should follow good practices. If you lose the private key then you will need to generate a new key pair.

Suppose you created your keys on your desktop, and you want to use them to ssh to your Linux home server without using a standard password. To do this you create an authorized keys file on the server:

```
// secure-copy your public key to the linux server (assuming the server is
named 'pi')
$ scp ~/.ssh/id_rsa.pub myname@pi:~/
The authenticity of host 'pi (192.168.1.90)' can't be established.
ECDSA key fingerprint is SHA256:iP...
ECDSA key fingerprint is MD5:79:54:...
Are you sure you want to continue connecting (yes/no)? yes
```

```
Warning: Permanently added 'pi,192.168.1.90' (ECDSA) to the list of known hosts.

// On the server create your authorized_keys file if it does not exist
// inside '~/.ssh', and then 'cat' your public key to the end of the file.
// The authorized_keys file should always be protected.

$ ssh myname@pi
myname@pi's password:
$ mkdir ~/.ssh
$ chmod 700 ~/.ssh
$ cd ~/.ssh
$ touch authorized_keys
$ chmod 600 authorized_keys
$ cat /path/to/id_rsa.pub >> authorized_keys
$ tail -1 authorized_keys
ssh-rsa AAAAB3...6oLYnLx5d myname@somewhere.com
$ rm /path/to/id_rsa.pub
// logout from the server session
$ exit

// Back on the desktop verify that you can ssh into the server using only
// the key's passphrase (and not your password):
$ ssh myname@pi
Enter passphrase for key '/home/myname/.ssh/id_rsa':
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.15.0-1027-raspi aarch64)
...
$ exit

// If you want to only allow ssh access to your account from a specific
// computer in your LAN, than limit the hosts which are allowed by using
// the 'from=' option.  Edit the authorized_keys file and prepend the
// entry like this:
$ pwd
/home/myname/.ssh
// allow from host with IP address 192.168.1.65, and from localhost:
$ nano authorized_keys
$ tail -1 authorized_keys
from="192.168.1.65" ssh-rsa AAAAB3...6oLYnLx5d myname@somewhere.com
```

Configure an SSH agent

The goal here is to start an SSH agent on your desktop, and add your key(s) to the agent.

With a few tweaks we allow other programs to inherit the ssh-agent *environment variables* and we avoid entering passwords and passphrases throughout the day, or until you logout, reboot or turn off your desktop.

Script to start an ssh-agent

Download the shell script named *prime-ssh-keys.sh* to start the agent. The script saves the environment variables in a file named:

> ~/.ssh-agent-info-YOUR-FULL-HOSTNAME

```
// Copy the shell script to your home 'bin' directory; create it if needed:
$ cd
$ mkdir bin
$ cp /path/to/prime-ssh-keys.sh ~/bin/
```

```
$ chmod 755 ~/bin/prime-ssh-keys.sh

// Run the script:
$ ~/bin/prime-ssh-keys.sh
Enter passphrase for /home/myname/.ssh/id_rsa:
Identity added: /home/myname/.ssh/id_rsa (/home/myname/.ssh/id_rsa)

// This file sets and exports environment variables for the socket and the PID:
$ cat ~/.ssh-agent-info-desktop.home
SSH_AUTH_SOCK=/tmp/ssh-XXXXXXxRlqsm/agent.21324; export SSH_AUTH_SOCK;
SSH_AGENT_PID=21325; export SSH_AGENT_PID;

// Now if you 'source' the agent file to inherit the environment variables
//  you will be able to ssh into the linux server without using
//  a password or passphrase:
$ . ~/.ssh-agent-info-desktop.home
$ ssh -Y pi.home
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.15.0-1027-raspi aarch64)
...
Last login: Thu May 11 23:56:17 2023 from desktop.home
$
```

Script to start an ssh-agent at initial login

In a MATE desktop setting, you can add startup programs that run as soon as you log into your desktop. You can find the startup options in:

Menus -> System -> Preferences -> Personal -> Startup Applications

Create and add the script – it will open a temporary terminal window asking for the passphrase(s) for your key(s). The terminal window can be any terminal the allows you to run a shell script as an argument. You can use *mate-terminal*, or if you have installed the **xterm** package then you can use *xterm*. Note that the script invokes a shell (*/bin/sh* is a symbolic link to */bin/bash*, and starts with a shebang)

```
// make the 'bin' directory if it does not exist:
$ cd
$ touch ~/bin/exec-prime-ssh-keys.sh
$ chmod 755 ~/bin/exec-prime-ssh-keys.sh
$ nano ~/bin/exec-prime-ssh-keys.sh
$ cat ~/bin/exec-prime-ssh-keys.sh
#!/bin/sh

#Decide which terminal command you will use:
exec mate-terminal -e /home/myname/bin/prime-ssh-keys.sh & 2>/dev/null
#exec xterm -u8 -e /home/myname/bin/prime-ssh-keys.sh & 2>/dev/null
```

Another useful tactic is to get your personal bash shell configuration file to inherit the SSH agent's environment variables. Create a small script named *~/.bash_ssh_env* which provides the variables. You can process that file in your *~/.bashrc* file so that any new terminal window you launch will always inherit the variables. As well, other scripts which might need the variables can do the same.

```
// First create .bash_ssh_env; we 'cat' it after to show what it contains:
$ nano ~/.bash_ssh_env
$ cat ~/.bash_ssh_env

ssh_info_file=$HOME/.ssh-agent-info-`/usr/bin/hostname`
if [ -f $ssh_info_file ] ; then
  . $ssh_info_file
```

```
fi

// Then source ~/.bash_ssh_env inside your .bashrc file by simply including
// the following line in ~/.bashrc:

. ~/.bash_ssh_env
```

Enabling Remote Desktop Services

In case you have various devices at home that you would like to use in a Linux desktop fashion, but you do not want to change the current environment on those devices, then you can configure your home Linux server to provide that opportunity.

For all remote desktop options, first create an *.Xsession* file in your home directory on the server and configure it to start a MATE desktop session:

```
$ cd
$ nano .Xsession
$ cat .Xsession
/usr/bin/mate-session
```

Configure Remote Desktop Protocol (RDP)

Most operating systems have client support for Microsoft's Remote Desktop Protocol. On Linux there is also a software package named *xrdp* which can provide RDP.

Note that RDP is typically not really secure without adding some additional security features. However, from within your home network xrdp is okay.

We install xrdp, tweak the configuration a little, and restart xrdp:

```
$ sudo apt install xrdp
...
The following additional packages will be installed:
  xorgxrdp
...
Setting up xrdp (0.9.17-2ubuntu2) ...

Generating 2048 bit rsa key...

ssl_gen_key_xrdp1 ok

saving to /etc/xrdp/rsakeys.ini

Created symlink /etc/systemd/system/multi-user.target.wants/xrdp-sesman.service ...
Created symlink /etc/systemd/system/multi-user.target.wants/xrdp.service ...
Setting up xorgxrdp (1:0.2.17-1build1) ...
...

// The configuration file is xrdp.ini; here we disable ipv6 by stipulating
// 'tcp://:3389' in the port configuration:
$ cd /etc/xrdp
$ sudo cp -p xrdp.ini xrdp.ini.orig
$ sudo nano xrdp.ini
$ diff xrdp.ini.orig xrdp.ini
```

```
23c23,24
< port=3389
---
> ;;port=3389
> port=tcp://:3389

// restart xrdp
$ sudo systemctl restart xrdp
```

You can test the setup from any other linux computer with *remmina* installed, or you can secure-shell into the Linux server with X11 forwarding using the '-Y' option and run 'remmina' from the command-line; that is:

```
// Suppose that your server is named pi.home
$ ssh -Y pi.home

// Test if X11 forwarding is working:
$ xhost
access control enabled, only authorized clients can connect
SI:localuser:myname


// If remmina is not installed, then install it:
$ sudo apt install remmina
$ remmina

// Start remmina and log into the server with your username and password.
// You can save a connection profile with a custom resolution setting
// to get the best possible presentation.
```

There are many web-based tutorials; check out *XRDP on Ubuntu 22.04*. This tutorial shows how to connect from Windows and from macOS as well.

Configure X2Go

X2Go is my favourite remote desktop setup since it runs seamlessly via secure shell connections and can use SSH key pairs to avoid password use. Thus I can comfortably connect as a remote desktop to remote servers across the continent, and write and test code as if I was down the hall from the remote server.

```
// install both the server and the client software
$ sudo apt install x2goserver x2goclient
...
Setting up x2goserver-x2goagent (4.1.0.3-5) ...
Setting up x2goserver (4.1.0.3-5) ...
Created symlink /etc/systemd/system/multi-user.target.wants/x2goserver.service
...

$ systemctl list-unit-files | grep -i x2go
x2goserver.service                          enabled         enabled
```

Your local desktop must have *x2goclient* installed so that you can start a remote X2Go desktop. From any Linux desktop running an X.org service you can start it from the command-line. This way the client inherits the SSH environment variables (you can also embed a small shell script which provides the environment in a custom application launcher).

```
// Bring up the application window; redirect stderr to /dev/null to ignore
//  various uninteresting messages.
$ x2goclient 2>/dev/null
```

Create and save the session parameters by starting a new session: 1. Session -> New Session a. Give the session a name b. Add the host name c. Add your login name d. Check auto login 2. Change tabs to the Input/Output tab a. Select a useful custom display geometry (e.g. Width: 1152 Height: 864) 3. Change tabs to the Media tab a. Disable sound and client-side printing 4. Click on the 'OK' button to save the session data 5. Start the client connection by clicking on the session name on the right

If you are using X2Go between different Linux varieties you might need to solve a few problems like font paths.

At the time of documenting this setup I found that starting the x2goclient from an older Linux system like CentOS 7 to this Ubuntu system needed a tweak in the Ubuntu Pi's Secure Shell /etc/ssh/sshd_config configuration; it was fixed by adding 'PubkeyAcceptedAlgorithms +ssh-rsa'. This was not needed between similar Ubuntu setups.

Starting Up an NFS Service for Other Linux Devices

Usually your home directory on your Linux desktop is where most of your important files reside. When you shutdown your desktop then those files are inaccessible. If you also have a Linux laptop then it would be nice to have the same home directory available on both the desktop and the laptop.

Putting your home directory on the Linux server would solve this issue. Here we look at installing an NFS service for 24x7 availability.

Configure the NFS Daemons

First install the needed packages. The *nfs-common* package contains both client and server elements; the *nfs-kernel-server* contains the server daemons *rpc.mountd* and *rpc.nfsd*.

```
$ sudo apt install nfs-common nfs-kernel-server
...
The following NEW packages will be installed:
  keyutils libevent-core-2.1-7 nfs-common nfs-kernel-server rpcbind
...
Setting up rpcbind (1.2.6-2build1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/rpcbind.service ...
Created symlink /etc/systemd/system/sockets.target.wants/rpcbind.socket ...
...
Creating config file /etc/idmapd.conf with new version
Creating config file /etc/nfs.conf with new version
...
Created symlink /etc/systemd/system/multi-user.target.wants/nfs-client.target ...
Created symlink /etc/systemd/system/remote-fs.target.wants/nfs-client.target ...
...
Setting up nfs-kernel-server (1:2.6.1-1ubuntu1.2) ...
Created symlink /etc/systemd/system/nfs-client.target.wants/nfs-blkmap.service ...
Created symlink /etc/systemd/system/multi-user.target.wants/nfs-server.service ...
...
Creating config file /etc/exports with new version
Creating config file /etc/default/nfs-kernel-server with new version
...

// show NFS and RPC services which are now enabled:
$ systemctl list-unit-files --state=enabled | egrep 'nfs|rpc'
nfs-blkmap.service                  enabled enabled
nfs-server.service                  enabled enabled
rpcbind.service                     enabled enabled
rpcbind.socket                      enabled enabled
nfs-client.target                   enabled enabled
```

```
// show NFS and RPC processes currently running:
$ ps -ef |egrep 'nfs|rpc'
\_rpc      271754       1  0 10:45 ?        00:00:00 /sbin/rpcbind -f -w
root      272176       2  0 10:46 ?        00:00:00 [rpciod]
root      272280       1  0 10:46 ?        00:00:00 /usr/sbin/rpc.idmapd
statd     272282       1  0 10:46 ?        00:00:00 /sbin/rpc.statd
root      272285       1  0 10:46 ?        00:00:00 /usr/sbin/nfsdcld
root      272286       1  0 10:46 ?        00:00:00 /usr/sbin/rpc.mountd
root      272294       2  0 10:46 ?        00:00:00 [nfsd]
root      272295       2  0 10:46 ?        00:00:00 [nfsd]
root      272296       2  0 10:46 ?        00:00:00 [nfsd]
root      272297       2  0 10:46 ?        00:00:00 [nfsd]
root      272298       2  0 10:46 ?        00:00:00 [nfsd]
root      272299       2  0 10:46 ?        00:00:00 [nfsd]
root      272300       2  0 10:46 ?        00:00:00 [nfsd]
root      272301       2  0 10:46 ?        00:00:00 [nfsd]
```

There are a few files to configure. We remove IPv6 RPC services by commenting out *udp6* and *tcp6* from /etc/netconfig:

```
$ sudo cp -p /etc/netconfig /etc/netconfig.orig
$ sudo nano /etc/netconfig
$ diff /etc/netconfig.orig /etc/netconfig
15,16c15,16
< udp6       tpi_clts      v     inet6    udp     -       -
< tcp6       tpi_cots_ord  v     inet6    tcp     -       -
---
> #udp6       tpi_clts      v     inet6    udp     -       -
> #tcp6       tpi_cots_ord  v     inet6    tcp     -       -
```

Before restarting the daemons we see various IPv6 processes running:

```
$ sudo lsof -i | grep rpc | grep IPv6
systemd        1      root  144u  IPv6 899098    0t0  TCP :sunrpc (LISTEN)
systemd        1      root  145u  IPv6 899100    0t0  UDP :sunrpc
rpcbind   271754      \_rpc    6u  IPv6 899098    0t0  TCP :sunrpc (LISTEN)
rpcbind   271754      \_rpc    7u  IPv6 899100    0t0  UDP :sunrpc
rpc.statd 272282     statd   10u  IPv6 897604    0t0  UDP :59540
rpc.statd 272282     statd   11u  IPv6 897608    0t0  TCP :47971 (LISTEN)
rpc.mount 272286      root    6u  IPv6 900420    0t0  UDP :53821
rpc.mount 272286      root    7u  IPv6 902332    0t0  TCP :45067 (LISTEN)
rpc.mount 272286      root   10u  IPv6 902347    0t0  UDP :43059
rpc.mount 272286      root   11u  IPv6 902352    0t0  TCP :39207 (LISTEN)
rpc.mount 272286      root   14u  IPv6 902367    0t0  UDP :52374
rpc.mount 272286      root   15u  IPv6 902372    0t0  TCP :52705 (LISTEN)

// restart the daemons (only the rpcbind ports still report IPv6)
$ sudo systemctl restart rpcbind nfs-server rpc-statd
```

Finally, the NFS shares need to be published and shared. It is best to export */home* so that logins on both the server and any clients use the same '/home' directories. If you use other directory names you need to change the home directory path in the password file, or create a symbolic link from */home* to the new directory – though we can do this, it is a bit messy.

```
// Edit the exports file on the NFS server:
$ sudo cp -p /etc/exports /etc/exports.orig
$ sudo nano /etc/exports
```

```
$ diff /etc/exports.orig /etc/exports
6c6,11
< #
---
>
> # Export home directories using NFSv3 syntax
> # Limit nfs access to the IP address of the client node(s)
> /home   192.168.1.65(rw,sync,no_subtree_check) \
>         192.168.1.85(rw,sync,no_subtree_check)
>
```

Then export the share. It can then be mounted on other clients.

```
$ sudo exportfs -a
```

Configure NFS Clients on Other Hosts

We will use *autofs* on other Linux hosts to automatically mount and unmount the home directory on demand. Note that any current directories in /home on your client will be affected, since we are going to mount the server's /home directory using the /home directory. Anything in that directory will be hidden until you unmount the server's version of /home. Consider migrating the content of /home to the server.

This example uses another Ubuntu host as the NFS client. We install autofs on it and then edit the automount map files as needed.

```
// install the autofs package
$ sudo apt install autofs
The following additional packages will be installed:
  keyutils libevent-core-2.1-7 nfs-common rpcbind
...
Setting up rpcbind (1.2.6-2build1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/rpcbind.service ...
Created symlink /etc/systemd/system/sockets.target.wants/rpcbind.socket ...
...
Setting up autofs (5.1.8-1ubuntu1.2) ...
Creating config file /etc/auto.master with new version
...
Setting up nfs-common (1:2.6.1-1ubuntu1.2) ...
...
Created symlink /etc/systemd/system/multi-user.target.wants/nfs-client.target ...
Created symlink /etc/systemd/system/remote-fs.target.wants/nfs-client.target ...

// If you are getting used to reading 'man' pages, then it is useful to use
// the 'man -k' option to get a list and description of all man pages available
// matching a topic:
$ man -k autofs
auto.master (5)      - Master Map for automounter consulted by autofs
autofs (5)           - Format of the automounter maps
autofs (8)           - Service control for the automounter
autofs.conf (5)      - autofs configuration
automount (8)        - manage autofs mount points

// Now edit configuration file /etc/autofs.conf and get rid of the 'amd' section
// Keep things simple.
$ cd /etc
$ sudo cp -p autofs.conf autofs.conf.orig
$ sudo nano autofs.conf
```

```
$ diff autofs.conf.orig autofs.conf
404c404
< [ amd ]
---
> #[ amd ]
410c410
< dismount_interval = 300
---
> #dismount_interval = 300

// Edit configuration file /etc/auto.master
// We comment out the '+auto.master' line and append the auto.home configuration

$ sudo cp -p auto.master auto.master.orig
$ sudo nano auto.master
$ diff auto.master.orig auto.master
22c22
< +dir:/etc/auto.master.d
---
> #+dir:/etc/auto.master.d
38c38,43
< +auto.master
---
> #+auto.master
>
> ## This is a direct mountpoint for NFS-provided home directories ..
> /-      /etc/auto.home --timeout 300
>

// We create /etc/auto.home
$ sudo nano /etc/auto.home
$ cat /etc/auto.home
## Every regular user you create needs an entry in this file
##
/home/myname \
  -rw,hard,intr,rsize=32768,wsize=32768,retrans=2,timeo=600,tcp,nfsvers=3 \
  192.168.1.90:/home/myname
```

Now, you would normally restart autofs - but do not do this immediately:

```
// restart autofs:
$ sudo systemctl restart autofs
```

Consider your impending ‘chicken-and-egg’ issue regarding being a regular user logged into your home directory while you restart autofs, causing your current home directory to disappear. There is some help on this topic in the appendix. Perhaps the easiest solution if you do not want to fiddle with changes to root access is to **reboot your client device**. Then when the client is back up again you should now have a home directory that is served by your server.

```
$ pwd
/home/myname
$ df -h .
Filesystem                    Size  Used Avail Use% Mounted on
192.168.1.90:/home/myname      50G  464M   50G   1% /home/myname
```

Topics To Document

These topics are not yet documented. This is a placeholder for now.

Creating a Web Site on the Server

description here.

Enabling a Git Service and Browsing It on Your Web Site

description here.

Other Possible Services

Other services will be added to this document in the future;
- some examples: bind DNS, MySQL, KVM virtualization, Ansible, ...

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 USB-based 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

This is a generic approach to creating an installation image on removable media; it generally works for various Linux distributions.

```
// 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, 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 not real; it is a place-holder for your 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).

Here are 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 * Once your new server is up and running you can further split your third partition into a fourth partition. You only need to unmount the third partition temporarily and use gparted to create another partition, make a file system on it, and alter /etc/fstab. I did this in order to move my home directory files to the fourth partition.

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

Here are some 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* filesystems 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, let's 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 – that is all.

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 with 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 backup 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
```

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
touch example	Creates a new (and empty) file named 'example'
rm -i example	Removes the file named 'example' if you respond with: y

Command	Purpose
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)

The last example in the above list:

env|sort|less

is actually 3 different commands *piped* together: - env - output the environment variables in your shell - sort - sort the incoming text, by default alphabetically - less - show the output a page at a time

A pipe, represented by a vertical line, sends the output from one command as the input for the next command. It is a hallmark of the UNIX way of doing things - create small programs that do one thing well, and string the commands together to accomplish a larger task.

A MATE Configuration Exercise

Here are a series of exercises you can try on a fresh installation of the MATE desktop. By default, the initial mate configuration has 2 panels (taskbars):

- the top panel has:
  - a global menu button on the left
  - on the right is a group of icons that represent the state of things, known as 'Indicator Applet Complete'
- the bottom panel has:
  - on the left: a 'show desktop' button
  - on the left: a window list area, where current open windows as shown
  - on the right: a workspace area with 4 workspaces
  - on the right: a trash can button

You may like this setup; but here is a small exercise to give you a quick start in making modifications.

Modify the Top Panel

1. Get rid of the bottom panel:
   - Right click on it and selecting 'Delete This Panel'. We will add some of its contents to the top panel.
2. On the top panel try a different menu presentation:
   - Right-click over the menu button on the top panel, and unlock its status. Then right-click and select 'Remove from Panel'.
   - Now right-click and select 'Add to Panel'. A list of applets mostly in alphabetical order will appear in a new window. Select 'Classic Menu' (or the 'Compact Menu') and click on 'Add' and it will appear on the panel. Right-click on it and move it completely to the left. Then right-click on it again and select 'Lock to Panel'. Leave the 'Add to Panel' window on your screen to continue adding applets.
3. Add a couple of separators:
   - Scroll down to find 'Separator' and add it to the panel. Then right-click on it, move it against the menu, and lock it to the panel. (Locked items cannot be accidentally moved; it is a mystery why it does not always prevent deletion of some applet buttons…)
   - Add another separator, and move it about an inch to the right of the first separator and lock it.
4. Add an active-window list:
   - Scroll down to find 'Window Selector' and add it to the panel to the right of the second separator. It is a bit tricky to select it to lock it. Right-click just to the right of the second separator line. If you see 'System Monitor' at the top of the menu list then lock it to the panel. Right click on it again and select

'Preferences'. In the 'Window Grouping' options select 'Group windows when space is limited' and then close that window.

5. Add a workspace switcher:
    - scroll down to find 'Workspace Switcher' and add it to the panel. Then right-click on it and select 'Preferences'. Reduce the number of workspaces to 2, and name them - for example rename one to 'Home' and the other to 'Projects'.
    - Now right-click and move it as far right as you can, and lock it.
6. Finally add a few of your own 'Launchers':
    - Add a firefox button:
        - right-click between the 2 separators and add to the panel: 'Application Launcher'. That will bring up a further selection. Click on Internet, then 'Firefox Web Browser' and add it. Again, right-click on the firefox button and move it to the left and lock it.
    - Add a terminal button:
        - right-click between the 2 separators and add to the panel: 'MATE Terminal' and again move and lock it.

Other Changes Done From the Control Center

1. Find the Control Center in the System Menu.
2. Under the 'Look and Feel' section select 'Screensaver':
    - Chanage the theme to something else, for example: 'Cosmos'.
    - Disable the 'lock screen' option if you wish.
3. Under the 'Look and Feel' section select 'Windows':
    - Change the 'Titlebar Action' to 'Roll up'
4. Under the 'Look and Feel' section select 'Appearance':
    - Try different themes
    - Try different backgrounds
5. Under the 'Personal' section select 'Startup Applications':
    - Disable 'Blueman Applet' and 'Power Manager'
    - Select 'Show hidden' and look at what is lurking underneath, disable things that clearly are not important to you.

Here are a Few Notes About Window Actions:

The 'Maximize Window' button (between the 'Minimize Window' button and the 'Close Window' on the right of each window) has difference actions depending on which mouse-click you use:

- a right-mouse-button-click over the maximize button maximizes the window horizontally. Another right-mouse-button-click will return it to the previous size.
- a middle-mouse-button-click over the maximize button maximizes the window vertically. Another middle-mouse-button-click will return it to the previous size.
- a left-mouse-button-click over the maximize button maximizes the window completely. Another left-mouse-button-click will return it to the previous size.

An Example Process and Script for Backups

There are many ways to do backups - this is just one example.
An example script in my github 'tools' area can do local system backups; it can also do external backups to a removable drive, such as an attached USB drive. If you do only local backups without creating a copy elsewhere then you run the risk of losing your data because of a major failure (like losing or overwriting the local disk) when you don't have another copy.

As well the script can handle large directories by using *rsync* to copy them them to a removable drive. Though 'rsync' is typically used for remote copies it can also be used locally.

The script also allows you to keep an additional copy of your large directories on the removable drive.

The example script requires a few configuration entries into a file named */etc/system-backup.conf*. You need a designated local directory; the files will be compressed so it requires only a few hundred megabytes per day for each day of the week. The provided example script also keeps the last week of each month for one year. If you use the external backup feature and/or the large directory backup feature then you simply need to provide the partition on the drive to use for backups, as well as the mounted name of the directories where the files will be copied.

In order to automate your backup script, you also need to create a *cronjob* which will automatically run your script in the time slot you pick. In the example below you:

```
* create the needed local directories specified in /etc/system-backup.conf
* edit the configuration file if you choose different directory names
* copy the configuration file and the script into place
* run the script in test mode to check configuration correctness
* create/edit the cronjob to run the local backup at 1 in the early morning
```

```
// Create local directory with permissions limiting access to
// backed-up files to users in the 'adm' group:
$ sudo mkdir /var/local-backups
$ sudo chown root:adm /var/local-backups
$ sudo mkdir /var/log/local-backups
$ sudo chown root:adm /var/local-backups

// Protect the backup directory by removing permissions for others:
$ sudo chmod o-rwx /var/local-backups

// Copy the configuration file to /etc/ and the shell script to /root/bin/
$ sudo cp /path/to/system-backup.conf /etc
$ sudo mkdir /root/bin
$ sudo cp /path/to/system-backup.sh /root/bin/
// The script must be marked as 'executable'; the chmod command will do that:
$ sudo chmod 755 /root/bin/system-backup.sh

// Edit the configuration file for the backups:
$ sudo nano /etc/system-backup.conf

// Run the script in debug mode from the command line to make sure
// that everything is correctly configured:
$ sudo /root/bin/system-backup.sh --test --local

// Create and edit the cronjob -- this example would run at 01:00 hrs
$ EDITOR=/bin/nano sudo crontab -e

// ( On Ubuntu 'crontab' puts cron-job files in /var/spool/cron/crontabs/ )
// Ask 'crontab' to list what that job is:
$ sudo crontab -l | tail -3

0 1 * * * /root/bin/system-backup.sh --local
```

If you will also synchronize backups to a USB drive, then you must make directories at the root of the USB filesystem for backups. The USB partition name and the names of the directories must match the configuration file setup. (example: your USB partition is /dev/sda1 and so you have set 'usbpartition' in the configuration file to 'sda1')

```
// Here we also prepare for doing large directory rsyncs as well:
$ sudo mount /dev/sda1 /mnt
$ cd /mnt
$ sudo mkdir backups rsyncs
$ sudo mkdir rsyncs/copy

// Be sure to unmount the drive
$ cd
```

```
$ sudo umount /mnt

// Run the script in debug mode from the command line to make sure
// that everything is correctly configured for external copies of your
// local backups (that is, the ones in /var/local-backups/):
$ sudo /root/bin/system-backup.sh --test --external

// If you will also do large directory backups then test that option
// as well.  You must have set 'largedirs' in the configuration file:
$ sudo /root/bin/system-backup.sh --test --rsync-large

// Finally edit the cronjob and add the external backup options to the command:
$ EDITOR=/bin/nano sudo crontab -e
$ sudo crontab -l | tail -3

0 1 * * * /root/bin/system-backup.sh --local --external --rsync-large
```

In case your USB drive is formated for Windows then it should be okay for all backups except for the large directories backup; that is, with the option '–rsync-large'.

The script might issue some warnings about trying to preserve LINUX permissions on the USB drive, but should otherwise work. I need to verify this case. You may have to change the rsync arguments in the script from *-aux* to *-rltux*. I need to test the Windows-formatted usb drive opton.

If you ever need to restore files from your backups then you should unpack the *tarballs* (compressed 'tar' files) on a Linux system and copy the needed files into place on the filesystem.

LAN (Local Area Network) Configuration files

There are some common networking files that are interesting to configure especially if you have more than one Linux computer on your home network. They are useful on your home Linux server as well, since it clarifies some configuration settings for some future services you might like to enable, for example, a web service.

You can only configure the hosts file if you have reserved IP addresses in your home router for your special devices, like your home Linux server.

The list of files that I like to manage are:

- /etc/networks
  - Read the man page on 'networks'
  - Add a local domain for your home network here. We will call this domain *.home*, that is, if your server's short name is *pi*, then its long name becomes *pi.home*
  - Avoid problems – do not use a valid Top Level Domain (TLD).
    '.home' is not a TLD (at least not yet..)
- /etc/hosts
  - Read the man page on 'hosts'
  - Add some IP addresses you have reserved in your home router for your hostnames
  - Be sure to include your home server

Your router's private network address is used when declaring your home domain; the private netowrk is typically something like 192.168.1.0 or 10.0.0.0. I did a quick analysis of a list of the IP addresses of common routers. The network address of the router is usually obtained by dropping the last octet from its IP address. The top 4 network addresses (without a trailing .0) were:

- 192.168.0
- 192.168.1
- 192.168.2
- 10.0.0

Note that your home network's IPv4 address space is always in the private network space – that is, network traffic addressed to devices in a private network is never routed over the Internet.

The Ubuntu version of the /etc/hosts file automatically puts your hostname within the 'localhost' network during installation; we will comment that out.

Here are the examples:

```
// /etc/networks
// You can look at your router's management web page to understand
// what your network address is - the example used here is: 192.168.1.0
// You can drop the last octet, that is the '.0', but I leave it in:

$ man networks
$ sudo cp -p /etc/networks /etc/networks.orig
$ sudo nano /etc/networks
$ cat /etc/networks
link-local 169.254.0.0
home  192.168.1.0

$ diff /etc/networks.orig /etc/networks
2a3
> home  192.168.1.0

// /etc/hosts
// Add your favourite devices with their reserved hostnames to /etc/hosts
// Each host gets it full name with .home attached, as well as its short name
// and any aliases:

$ man hosts
$ sudo cp -p /etc/hosts /etc/hosts.orig
$ sudo nano /etc/hosts
$ # diff /etc/hosts.orig /etc/hosts
2c2
< 127.0.1.1     pi
---
> #127.0.1.1    pi
9a10,13
>
> 192.168.1.90  pi.home pi www
> 192.168.1.65  desktop.home desktop
> 192.168.1.80  odroid.home odroid
> 192.168.1.81  laptop.home laptop
> 192.168.1.86  inkjet.home inkjet
```

Changing the Server's Hostname

Now that we have a *.home* domain we can rename our official server's hostname. Suppose the server was originally named *pi* during the installation:

```
// look at what you set your hostname to during the installation:
$ hostname
pi

// Give the server a fully-qualified hostname using 'hostnamectl'
// Note that older versions of hostnamectl required:
//      sudo hostnamectl set-hostname pi.home
$ sudo hostnamectl hostname pi.home
```

```
$ hostname
pi.home
```

The Resolver and Looking Up Your Local Hostnames

Well-known traditional command-line tools for querying DNS [4] are:

- host
- nslookup
- dig

These tools look in */etc/resolv.conf* for the nameserver(s) to query when looking up IP addresses or hostnames.

Of course, external DNS servers know nothing about your private local network. So, when you try querying a private host using one of the above utilities, you might see:

```
$ host pi.home
pi.home has address 192.168.1.90
Host pi.home not found: 3(NXDOMAIN)
```

The 'host' command looked at the /etc/hosts file, but might also consulted the listed nameservers. This is because it consults an important file named */etc/nsswitch.conf* which configures the order to try when looking up host and other data. Because 'files' is first, the daemon consults /etc/hosts before doing any dns request:

```
$ grep hosts /etc/nsswitch.conf
hosts:          files mdns4_minimal [NOTFOUND=return] dns
```

There is another command-line tool – *getent* – for looking up local dns data, and does not consult nameservers:

```
$ getent hosts pi
192.168.1.90    pi.home pi web
$ getent hosts 192.168.1.90
192.168.1.90    pi.home pi
```

Contemporary Linux version's using systemd-resolvd handle this case better since it acts as a local nameserver that handles local DNS lookups, especially local IP address lookups. However it still whines about hostname looks, though it returns the correct local lookup anyway:

```
// Try from another ubuntu host:

$ hostname
ubuntu.home

$ host pi
pi has address 192.168.1.90
Host pi not found: 3(NXDOMAIN)
$ echo $?
1

$ host 192.168.1.90
90.1.168.192.in-addr.arpa domain name pointer pi.


// Now from the pi host - NOTE that it will not emit an NXDOMAIN message
// for its own hostname, and thus will return success, which is '0':

$ hostname
pi.home
```

[4] Domain Name System – how we look up hostnames

```
$ host pi
pi has address 192.168.1.90
$ echo $?
0
```

Modifying the Resolver's List of Nameservers

The resolver file would typically be created at bootup when your computer makes a DHCP request to the home router asking for an IP address and the names of the router's configured DNS servers.

On contemporary Linux versions the resolver file is created and managed differently than in older Linux versions. Recent Ubuntu LTS versions use a systemd service named *systemd-resolved* which by default manages the resolver file, and it runs a local DNS server:

```
// list open network connections and find a name match for 'resolve'
# lsof -i -P -n +c0 | grep resolve
systemd-resolve  568 systemd-resolve   13u  IPv4  20701      0t0  UDP 127.0.0.53:53
systemd-resolve  568 systemd-resolve   14u  IPv4  20702      0t0  TCP 127.0.0.53:53 (LISTEN)

// This is what the resolver file looks like on a newly installed Ubuntu node
$ tail -3 /etc/resolv.conf
nameserver 127.0.0.53
options edns0 trust-ad
search .

// the resolver file is actually a symbolic link into territory owned by systemd
$ ls -l /etc/resolv.conf
lrwxrwxrwx 1 root ... Mar 17 14:38 /etc/resolv.conf -> ../run/systemd/resolve/stub-resolv.conf

// get the resolver's status -- in this example the first DNS server is
// my router's IP address
$ resolvectl status
Global
       Protocols: -LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
resolv.conf mode: stub

Link 2 (ens3)
    Current Scopes: DNS
         Protocols: +DefaultRoute +LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
Current DNS Server: 192.168.1.254
       DNS Servers: 192.168.1.254 75.153.171.67
```

Sometimes you want to modify the list of external DNS servers – for example – the DNS servers used by *my* router have 'slow' days, so I like to have control over the list of DNS servers to fix this issue. Here is a look at the process.

Create a local copy of the resolver file - do not pollute systemd space:

```
// Remove the current resolver file (we don't want to edit systemd's file).
// This just removes the symbolic link:
$ sudo rm /etc/resolv.conf

// Get a copy of /usr/lib/systemd/resolv.conf for manual control of the resolver
$ sudo cp -p /usr/lib/systemd/resolv.conf /etc/resolv.conf

// Change only the comments at the top to show it is locally managed:
$ sudo nano /etc/resolv.conf

$ tail -3 /etc/resolv.conf
```

```
nameserver 127.0.0.53
options edns0 trust-ad
search .
```

By default this version of Ubuntu uses NetworkManager for network configuration, and the local systemd-resolved for DNS service. To make a permanent change to the DNS information known to systemd we configure NetworkManager using its management utility *nmcli*:

```
$ nmcli connection show
NAME                UUID                                  TYPE      DEVICE
Wired connection 1  91591311-3c9a-3541-8176-29a8b639fffa  ethernet  eth0
MY-SSID             924de702-7f7e-4e31-8dff-4bc968148f2b  wifi      --

$ nmcli connection show 'Wired connection 1' | grep -i dns
connection.mdns:                        -1 (default)
connection.dns-over-tls:                -1 (default)
ipv4.dns:                               --
ipv4.dns-search:                        --
ipv4.dns-options:                       --
...
IP4.DNS[1]:                             192.168.1.254
IP4.DNS[2]:                             75.153.171.67
```

Now we add some other DNS servers to this configuration using nmcli; it should persist after a reboot. We are adding well-known public IP addresses from Cloudflare (1.1.1.1), and from Google (8.8.8.8):

<– !! I need to check if we need to restart the network.. –>

```
$ sudo nmcli connection modify 'Wired connection 1' ipv4.dns "1.1.1.1,8.8.8.8"
$ nmcli connection show 'Wired connection 1' | grep -i dns
connection.mdns:                        -1 (default)
connection.dns-over-tls:                -1 (default)
ipv4.dns:                               1.1.1.1,8.8.8.8
ipv4.dns-search:                        --
...
IP4.DNS[1]:                             1.1.1.1
IP4.DNS[2]:                             8.8.8.8
IP4.DNS[3]:                             192.168.1.254
IP4.DNS[4]:                             75.153.171.67

// Check it with resolvectl:
$ resolvectl status|grep 'DNS Serv'
Current DNS Server: 1.1.1.1
       DNS Servers: 1.1.1.1 8.8.8.8 192.168.1.254 75.153.171.67
```

If ever you want to make a temporary change, use 'resolvctl' to do that; it will not persist after a reboot:

```
// Here the Pi's ethernet device name is 'eth0'
$ sudo resolvectl dns eth0 9.9.9.9 8.8.4.4 75.153.171.67

$ resolvectl status
Global
       Protocols: -LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
resolv.conf mode: foreign
      DNS Domain: ~.

Link 2 (eth0)
    Current Scopes: DNS
         Protocols: +DefaultRoute +LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
```

```
Current DNS Server: 9.9.9.9
       DNS Servers: 9.9.9.9 8.8.4.4 75.153.171.67
```

Other Ways of Becoming the Superuser in a Restricted Environment

You might run into a chicken-and-egg problem occasionally because you need to do something like:

- move your home directory files
- fix a problem with logging in as a regular user
- you accidentally removed the sudo package

Then you realize that you need to be logged in as a regular user to sudo to *root*, but you cannot be logged in to accomplish your task.

Setting a Password for the Superuser

One solution is to set a password for the *root* user - you can always disable the password afterwards.

This is an example of setting a password for root – as always you set a **strong** password:

```
// Normally password access is locked in /etc/shadow - we unlock it when
// we set a password:
$ man passwd
$ man 5 passwd
$ sudo /bin/bash
# id
uid=0(root) gid=0(root) groups=0(root)
# head -1 /etc/shadow
root:!:19476:0:99999:7:::
# passwd
New password:
Retype new password:
passwd: password updated successfully

// We now see an encrypted password in the password field in the shadow file:
# head -1 /etc/shadow
root:$y$j9T$FFNwo6b8WAoEu...tQQhPaSIumPjNPXjWAe7h2M4:19519:0:99999:7:::
```

Now we can log in directly as root at the server's text console; remember that it is foolish to login as 'root' to a graphical environment. Using web browers as 'root' is not smart.

To lock the root user from using a password, use the **-l** option; you can unlock it in the future with the **-u** option.

```
// lock it:
# passwd -l root
passwd: password expiry information changed.

# head -1 /etc/shadow
root:!$y$j9T$FFNwo6b8WAoEu...tQQhPaSIumPjNPXjWAe7h2M4:19519:0:99999:7:::
```

Allowing Secure-shell Access From Another Device in Your LAN

Another solution is to allow another device in your home network to have secure-shell access to the root account on your Ubuntu server or desktop. It is mentioned briefly in the secure-shell section of this guide, but I summarize the main options here.

The secure-shell daemon must be installed and running on the target device. In */etc/ssh/sshd_config* add the remote device's IP address with *root@* prefixing it to the 'AllowUsers' rule. Here access to the root account is allowed from 192.168.1.65:

```
# id
uid=0(root) gid=0(root) groups=0(root)
# grep AllowUsers /etc/ssh/sshd_config
AllowUsers      myname@192.168.1.* root@192.168.1.65 *@localhost
```

Then root's */root/.ssh/authorized_keys* file must specifically allow the remote device; in this case we use the *from="* option (see the man page for 'authorized_keys'). As well, if you also allow localhost (127.0.0.1) you would allow your login account to ssh locally to root:

```
# tail -1 ~/.ssh/authorized_keys
from="127.0.0.1,192.168.1.65" ssh-rsa AAAAB3...6oLYnLx5d myname@somewhere.com

$ whoami
myname

$ ssh -A -Y root@localhost
Last login: Tue Jun 13 15:10:42 2023 from desktop.home
# ps -ef --forest|grep ssh
root       685       1  May26 ?      00:00:00 sshd: /usr/sbin/sshd -D [listener] ...
root    395328     685  09:27 ?      00:00:00  \_ sshd: myname [priv]
myname  395330  395328  09:27 ?      00:00:00  |   \_ sshd: myname@pts/0
myname  395414  395331  09:29 pts/0  00:00:00  |           \_ ssh -A -Y root@localhost
root    395415     685  09:29 ?      00:00:00  \_ sshd: root@pts/1
root    395460  395429  09:30 pts/1  00:00:00          \_ grep --color=auto ssh
myname    3693       1  May26 ?      00:00:00 ssh-agent
```