

# Lab 1

## Installation of the OS

### 1.1 Objectives

The goal of this first session is to install a Debian/Linux operating system from scratch on a Intel x86-based computer. The installation will be made on a removable USB disk. More information about installing Linux can be found in [1, 2]. A detailed description of the Linux boot process can be found in [3]

At the end of installation you should be able to use the installed operating system, i.e., doing a correct boot and login process.

### 1.2 Before you start

- Review UNIX basic commands: `cd`, `ls`, ...
- Basic usage of the `vi` editor

The rest of the session assumes the user connected as superuser, to become superuser we run the following command:

```
$ sudo su
```

When having to specify a command prompt, a convention we use for all the labs in this course is to use the symbol `$` as a command that needs to be run as a regular user, and `#` for commands that require superuser privileges.

### 1.3 Installation

#### 1.3.1 Hardware Identification

There are many graphical tools that allow to identify, list, and provide details about the hardware present on a system. However, during this first session we will provide some low level tools and commands that may be of assistance when no graphical interface is present on the system. We will mostly focus on the identification and understanding of the hard drive topology.

First we will identify the different hardware components present on the system. Hence we run:

```
# lspci
...
# lsusb
```

These commands allow to identify the devices directly connected to PCI and USB buses. Look at the manpage for each command to see the accepted parameters. Now try to fill as many entries as you can in Table 1.1.

Part	Hardware model	Device Name
Network Card		
Internal Hard Drive		
USB Hard Drive		

Table 1.1: Hardware table

Now, we will try to map the found hardware parts with the actual devices, plus try to find the devices not found using the previous commands. To do so we run:

```
# dmesg
```

Now you should be able to finish filling up the table. Remember that devices are generally stored into the `/dev` directory. Consider also that harddrives connected to the SATA bus follow the naming convention of `sd` (as of SATA<sup>1</sup> Disk), while sorting the drive names alphabetically: `a` for the first disk, `b` for the second, ... down to `zz`. Consequently, the third SATA harddrive on a system will have the device name of `/dev/sdc`. For more information about harddrive naming conventions you can refer to [4]

### 1.3.2 Disk configuration: partitions

The first step to install the system is to partition the disk on the USB drive. First step is to find which is the device corresponding to the USB drive, it is in the form: `/dev/usb2`.

An important point before continuing, we are using Ubuntu as our temporary system. Ubuntu automatically mounts all external drives when plugged in, this may lead to issues later during the lab, as the disk already has some partitions. It is **strongly recommended** to unmount all the external drives before continuing. To do so run:

```
# umount /dev/usb*
```

Now, to partition the disk use the `gdisk` command on the device `/dev/usb`. With this command you can:

- Determine the number of sectors of your disk, and its size.
- Erase the current partition table (it is rather important to do this so you start from a clean slate). This may be achieved with the `o` command.
- Create the partitions indicated in Table 1.2 (select the appropriate size for each one). Since we will be using a GPT partition table, remember that in this case we can create up to 128 partitions without any other restrictions.
- Write the contents of the partition table to disk (remember to do so before leaving from `gdisk` command).

Check that the device files corresponding to the new partitions appear in the `/dev` directory.

 Write here the full names of such new files

Don't forget to save the changes before exiting `gdisk`.

<sup>1</sup>Actually it stands for SCSI disk, but since it is being superseded by SATA the same naming convention applies

<sup>2</sup>Be careful, `usb` actually is not the device name, find the proper one understanding the naming conventions stated above

Device	Code	Size	Mount-point	Comments
/dev/usb1	EFI System Partition (ESP) (EF00)	512MB	/boot/efi	This partition will hold the necessary EFI Boot information. It MUST be formatted with FAT32 (vfat) filesystem
/dev/usb2	Linux (8304)	30GB+	/	This partition will hold the main system, a typical Debian installation requires around 5GB, however, when we add more software it may grow considerably. Use at least 30Gb
/dev/usb3	Linux (8300)	5GB	/usr/local	We don't use too much this partition during the course, with 5GB it should be enough, in a real system it depends on the actual requirements regarding self-compiled applications
/dev/usb4	Linux (8302)	100GB+	/home	This one in general is where most space should be devoted. You can put 100GB or more
/dev/usb5	Swap (8200)	2xRAM		Put twice the size of the machine's RAM <sup>1</sup>
/dev/usb6	Linux (8300)	20GB		Reserved for future use, you don't need to create it now

<sup>1</sup> Check how much the machine has with the `free` command

Table 1.2: Partition table

### 1.3.3 Disk configuration: Creating the file systems

Once you have created the necessary partitions, you must initialize the file system on those partitions that will contain your files and prepare the swap area for use (this step is not actually mandatory but recommended on embedded systems).

To format the swap area should use the command:

```
# mkswap device
```

To create a Linux file system in the other partitions, we use the command:

```
# mkfs -t fstype device
```

For each partition where you initialize a filesystem, where `fstype` can be: `ext4`, `btrfs`, `vfat`, `reiserfs`, ... as the type of filesystem you want to create on the partition. Currently the de facto standard in Linux systems is `ext4`, use this one when creating the filesystems. Be careful and create a `vfat` filesystem for `/dev/usb1`.

Depending on the type of file system that you want to use, the options to create the file system can be different. Browse through the `mkfs` man page to see the options which can be used.

Now give format to the partitions you made earlier given the indications you have on the table.

### 1.3.4 Mounting the filesystems to install the system

Now we have to mount the filesystems in a new temporary directory to be able to install the software. Let's create the mount points for the installation, using a new directory:

```
# mkdir /linux
```

and now we will mount all filesystems inside such a mount point (`/linux`), creating the appropriate directories inside, at the same time. Don't forget `/boot/efi` as it will be required to boot the new system.

```
# mount partition directory
```

Table 1.2 shows in which mount point (directory) each partition has to be mounted (remember, always inside `/linux`; for instance, `/` partition into `/linux`, `/home` partition into `/linux/home`, etc.). It is **important** to create the directories after mounting `/linux` but **before** mounting the rest.

### 1.3.5 Installation of the base system

Once you have prepared the partitions, our next step is to install the base operating system. This process may vary depending on the system. Usually Linux distributions are organized into packages, and the software installer decompresses them into the destination directory, and then automatically configures them (maybe with some hints from the user).

In our case, we will install from a prepackaged system image that is in the ASO SFTP server:

```
asoserver.pc.ac.upc.edu
```

You will have to decompress it into the USB disk. `cd` into the filesystem that will become the root (`/`) of your new installation (remember, mounted on `/linux`):

```
# cd /linux
```

And now get the image:

```
# sftp aso@asoserver.pc.ac.upc.edu
```

Use the following password: **AsORoCkSHaRd!**

From there you can download (using the command `get`) the file `/packages/aso-install.tar.gz`. To finally `untar` the file:

```
# tar xzf aso-install.tar.gz
```

Now look at the contents of the `/linux` directory. You should see that it has been populated with the basic components of your future system.

Optionally we can erase the downloaded file by running:

```
# rm aso-install.tar.gz
```

### 1.3.6 Finish mounting auxiliary systems

We also have to **bind**-mount the directories `/dev`, `/sys`, and `/proc` inside `/linux`, to temporarily expose the current existing devices to the new system. To do that, use the following command:

```
# for i in /dev /dev/pts /proc /sys /run; do
>     mount -B $i /linux/$i
> done
```


Try to understand what it does, since we will be using the `for` command several times during the course. Yay!, you just executed your first *script*!

Use the `mount` command without any parameters to see which filesystems are mounted, and verify that all USB disk partitions are mounted correctly in the appropriate directories, including the `/dev` directory with the system device files.



What is the purpose of the flag `-B` in the `mount` command?

## 1.4 Basic system configuration

 **Important note:** All the directory paths in this section refer to the future system, consequently, when we specify for example `/etc` we actually mean `/linux/etc`. Before re-booting, you should perform some more steps: configure system mountpoints through the `/etc/fstab` file and install a boot loader.

The configuration files in operating systems based on Unix/Linux are on the default directory `/etc` and almost always in text format. The Linux environments have many tools for text processing from command line (`grep`, `sed`, `tail`, `cut` ...) and editors (`vi`, `nano`, `emacs`, ...). During the installation process, only the `'vi'` editor will be available in most cases.

### 1.4.1 Configuring the file systems table (/etc/fstab)

In order for the file systems to be mounted correctly at system power-on, you need to generate a correct /etc/fstab file. To do this edit the file and make sure that the following parts exist:

- Add your swap partition:  
device none swap defaults 0 0
- Add root partition:  
device / ext4 defaults 0 1
- Add the rest of filesystems you created previously:  
device mountpoint fstype defaults 0 2

**?** Why /proc and /sys do not have any device attached?

### 1.4.2 Changing root directory

At this point, you can change the root directory of your system, and temporarily use the software that you installed in the system, instead of the one currently available on the system. To change the root of your system, use:

```
# chroot /linux
```

 **Note:** It is quite important to understand what the `chroot` command does.

From this point on, you can use the system we have installed, and access for example manual pages with the command `man`.

### 1.4.3 Configuring the keyboard

In Debian, the keyboard layout is configured by running:

```
# dpkg-reconfigure locales
# dpkg-reconfigure console-data
# dpkg-reconfigure keyboard-configuration
```

In the `locales` configuration make sure you select a suitable locale for your machine, by searching for it and pressing `<Spacebar>` to select it.

### 1.4.4 Configuring the boot process

Formerly, the operating system was installed on a partition that was marked as bootable in the partition table. The BIOS was searching for it, and then booting the system. This meant that we could have only one system working on a PC, and that if we wanted to boot from another partition, we should change the partition table and reboot. To address this limitation, second-level boot managers boot (bootstrap loaders) help in booting several operating systems. A boot loader is a set of programs residents in the disk drive, that allow the user to load other operating systems (including from other disk drives). They do the same as BIOS does with them: loading the OS into memory and transferring control. Among the most used we find: LILLO (Linux Loader), GRUB and NTLDR (used in systems from Microsoft).

Currently, we have the system installed, but we need to somehow point where our OS is to the UEFI Boot system so that next time you boot the computer it is done properly. To this end we will install the *GRUB* boot manager.

To configure correctly the boot of the machine using GRUB, we need to execute the script provided by Linux (remember to keep your root (/) directory set to /linux):

```
# grub-install --target=x86_64-efi /dev/usb
```

This script prepares the /boot directory in order to contain the information needed to boot the machine. The steps it performs (you do not need to repeat them) are:

- Copy the files needed for *GRUB* to /boot.
- Installs the bootloader in the GPT of the USB disk.

Besides the boot configuration, the system must inform to *GRUB* which kernel must be used to properly boot, this can be accomplished by editing the file /boot/grub/grub.cfg. Since this may be an involved process, Debian comes with system tools that allow the automatic creation of such file. In order to automatically update the /boot/grub/grub.cfg file to contain the UUID of the particular partition you are using it is necessary to run the script:

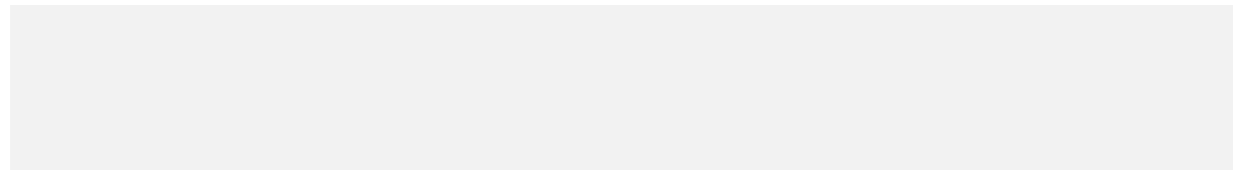
```
# update-grub
```

### 1.4.5 Setting up the passwords

In the current system, the existing user passwords are set to defaults, which is not what we want. To change the password we need to update the file /etc/shadow, look at it. As expected the file has hashed passwords rather than plain text. In order to be able to change the passwords we can use the command passwd.



Now change the passwords for the user *aso* and the user *root*

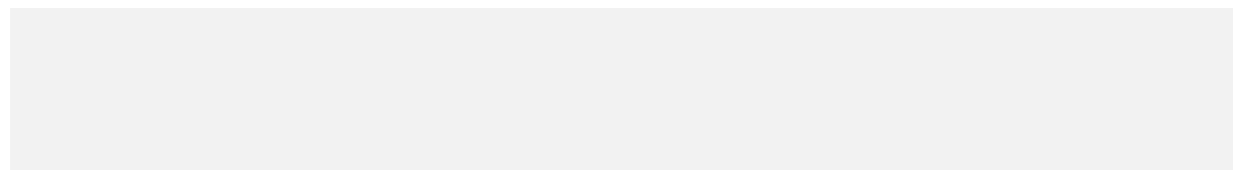


You can now exit the chroot shell.

Unmount all filesystems by running the relevant umount commands, and reboot using the shutdown or poweroff commands.



How have you executed the *shutdown* command?



## 1.5 Post-configuration

Boot the newly installed system. Remember, if you are on the Lab when the HP screen appears press **F9** to popup the Boot menú from the BIOS.



It is important to **select the Toshiba harddrive from the UEFI boot option**. Don't select the Legacy boot option, as it will try to boot with the former MBR mechanism which we didn't configure.

Once booted, if all went ok you should obtain a login prompt on a black screen. Now you have to perform a series of system configuration tasks on this new system. The system has two user accounts: root and aso. Login as aso using the password you set previously. In general you should use always

an unprivileged user to minimize the possibility of damaging your system by mistake. When you need to have user privileges (i.e. become root), use the command `su`:

```
$ su
# privileged-command      ← this will run as root
# exit
```

or

```
# su -c "privileged-command"
```

Nevertheless, `su` is rather cumbersome, it is better to install `sudo`. We will tackle this later in this session since we need the network for that.

### 1.5.1 Configuring filesystems

In file systems `ext3` and `ext4` we find certain properties that can be changed after formatting, with the `tune2fs` command. Using this command, change the frequency of checks of the filesystem in the `usb2` partition to 28 days.

**?** *What other parameters can be adjusted with the `tune2fs` command?*

### 1.5.2 Configuring system login messages

There are several configuration files that handle messages that appear during the process of login into the system. We want to change some of these messages.

**?** *Where are configuration files commonly located?*

Before the login prompt `asoclient login:`, there is a message similar to "Debian GNU/Linux 11". Often, we want to change this message. Now we want to change it for something like this (usually a message indicating that normal users activities can be recorded for security reasons):

```
#####
# This system is for the use of authorized users only.          #
# Individuals using this computer system without authority, or in #
# excess of their authority, are subject to having all of their  #
# activities on this system monitored and recorded by system    #
# personnel.                                                    #
#                                                                #
# In the course of monitoring individuals improperly using this  #
# system, or in the course of system maintenance, the activities #
# of authorized users may also be monitored.                    #
#                                                                #
# Anyone using this system expressly consents to such monitoring #
# and is advised that if such monitoring reveals possible       #
# evidence of criminal activity, system personnel may provide the #
# evidence of such monitoring to law enforcement officials.     #
#####
```

**?** *Which file should contain this message? And which command did you use to find the file?*



**Hint:** search for the file with the original contents that you want to replace. And please don't just copy ALL the above message, you can make it shorter!

After login, we obtain another message, the MOTD (or Message-Of-The-Day). Usually, this is used to give last-minute information to the users about the status of the system (for instance, contact information or system news).

Locate such file, and change it to report on how to contact with the system administrators.



*Which file have you modified?*

### 1.5.3 Network configuration

The next step in the lab session is to configure the network. This means that once this stage is complete, your system should be capable of communicate with other systems via the IP protocol. First, we will do the network configuration by hand, later we will use DHCP to configure it permanently.

Before starting, and to avoid mistakes we will flush the current network status, this may be achieved by running:

```
# ip link set dev <ethernet IF> downa
# ip link set dev <ethernet IF> up
```

<sup>a</sup>Change <ethernet IF> by the actual name of the interface. Try: \$ ip link show

#### Manual configuration

The manual network setup usually involves three steps:

1. Configuring the network interface using the `ip address` command
2. Configuring the routing table using `ip route` command
3. Setup name resolution in `/etc/resolv.conf` using `vi`

Check the manpages that correspond to these commands and files.



*Which interfaces are configured in your system?*

To configure properly the network, have in mind the data of Table 1.3.

Configure the Ethernet interface corresponding to the Gigabit Ethernet.

Ask the professor in the class for the IP address corresponding to your computer.



*Which command do you use to configure the IP address?*



Parameter	Configuration value
IP address	
Mask	
Gateway	
DNS Server	147.83.41.104

Table 1.3: Network information

Now you have to add the default gateway to the routing table.


 Which command do you use?



**Hint:** OK, this one may be complicated, the form of the command is: `# ip route [operation] default via [gateway]`. Don't forget to change `operation` and `gateway` by the proper values.

Finally, create the file `/etc/resolv.conf` with the DNS information. The Name Server if you are physically in the lab is 147.83.41.104.

Now, to check if the network is working we can `ping` any host on the Internet. Sadly, in the lab, for security reasons `ping` is not allowed. However, if you `ping` a particular host using its name, `ping` will try to resolve it on the DNS server, if the name resolution work, then this means the network is properly configured.

 Specify the `ping` command you used.

### Permanent configuration

Now we want the network to be configured properly at boot time and do not have to do it manually each time. First, delete the manual IP address running:

```
# ip address del 10.10.41.???/24 dev <ethernet IF>
```

Use `$ ip address show` to verify that the IP no longer is assigned to the interface.

On Debian (and other systems) the network configuration resides in several files in the `/etc/network` directory.

In the directory `/etc/network` there is a file `interfaces` that is where you configure different interfaces. Right now there is only configured the loopback interface. Add an entry in the `interfaces` file to

configure your network interface with the parameters you used previously. First add a line to indicate that we want to activate the interface automatically at boot:

```
auto <ethernet IF>
```

After indicating that, we will give all the necessary parameters to configure the interface:

```
iface <ethernet IF> inet static
```

And immediately all the necessary parameters (except the DNS server):

```
address 10.10.41.???  
network 10.10.41.0  
netmask 255.255.255.0  
gateway 10.10.41.1
```

Now, to check that we have configured it correctly, we could do a reboot. But in fact we do not need that. We can use the commands

```
# ifup <ethernet IF>  
# ifdown <ethernet IF>
```

to tell the system to reconfigure an interface according to the interfaces file.

Once you have made it work, now we want to obtain the network settings the system automatically using DHCP. Check the manual interfaces file (**man interfaces**) and use DHCP to configure <ethernet IF>.

We recommend, before changing the file, that you bring the network back down, to avoid inconsistencies, you can do that by running

```
# ifdown <ethernet IF>
```



*What changes you made to /etc/network/interfaces?*



**Hint:** if someday you want to configure DHCP without having to edit the file, or you just want to do it quick, you can run:

```
# dhclient <ethernet IF>
```

This will disappear on boot, but gives a neat and quick way to configure the interface.

## 1.6 Some final configuration

Since using `su` is a little painful, in *Lab 4* we will study a better tool, which is `sudo`, for now, and out of simplicity, we will just install `sudo` and configure the user `aso` to use it. In *Lab 2* we will explain in detail how software installation works, for now just run the following commands:

```
# apt update  
# apt install sudo
```

Now we add `aso` to the `sudo` group, this is achieved by running:

```
# usermod -a -G sudo aso
```

The command should be self explanatory, if you have doubts you can run:

```
$ man usermod
```

To visit the command manpage.

For this change to take effect, you have to close the session and log back in.

## 1.7 systemd and systemctl

### 1.7.1 Init process

The init process is the first process created during booting Unix operating systems. Init is typically assigned `pid=1` and it keeps running as a daemon until shutting down the OS. It is the root node of the process hierarchy and adopts all orphan processes (the child processes of the died processes). The init system is responsible for starting the userland daemons and services selected by the sysadmin (secure shell, web server, display manager, ...), creating sockets, mounting filesystems, initializing new hardware, ...

There are two main alternatives for the init system: the BSD-style (based on a script at `/etc/rc` directory) and the SystemV-style (`SysVinit`, based on `runlevels`). One of their drawbacks is their serial nature, that may result in a long boot time.

Several alternatives have been proposed to replace the previous init systems: `systemd`, `upstart`, `s6`, `Launchd`, ... We focus on `systemd`. Although some criticisms [5], `systemd` has been adopted by many Linux distributions.

`systemctl` is the command that allows the sysadmin to interact with `systemd` (`list/start/stop/restart/-configure services, ...`).

### 1.7.2 systemd concepts

- **units:** A *unit* represents a resource handled by `systemd`. We can imagine units as a services (`ssh,...`), mount points, sockets, .... `systemd` supports 11 unit types (service, socket, target, device, mount, automount, timer, swap, path, slice and scope).
- **unit file:** each unit is described by a plain text configuration file, the *unit file*, which name ends with its unit type (for instance, `ssh.service` or `boot.mount`). Unit files are divided into sections (labelled as `[Unit]`, `[Install]`, `[Service]`, `[Socket]`, ...) and, inside each section, the appropriate directives are initialized (`DirectiveName = value`).
- **unit state:** at boot time, units are *loaded* into memory. Most of them become *active*, waiting for the occurrence of a precise event (for instance, opening a socket, plugging a device, ...) that will trigger daemon's response.
- **dependencies:** A unit may require other units to be loaded simultaneously. In the unit file, four directives in the `[Init]` section specify the dependencies between units: `Want`, `Require`, `After` and `Before`.

### 1.7.3 systemctl basic operations

`systemctl` is the command that controls the `systemd` system and service manager. Its command line expects a `systemctl-command` and, optionally, flags and unit(s) name(s). This section describes some of the more usual `systemctl` commands.



**Hint:** TAB key autocompletes or suggests how to complete `systemctl` command lines.

The most common operation of `systemctl` is listing the units managed by `systemd`: `list-units` command offers this information. Note that, by default, `systemctl` results are paged using `less` command; press `h` to get a summary of `less` commands.

- `systemctl list-units` ⇒ list loaded active units
- `systemctl list-units --all` ⇒ list all loaded units (both active and inactive)
- `systemctl list-units --type type` ⇒ list loaded active units of a particular type (for instance, try with `service`).
- `systemctl list-units --failed` ⇒ lists units in failed state

Another common operation is obtaining more information about a particular unit. Several `systemctl` commands do this work:

- `systemctl status unit` ⇒ shows the status of the unit and its most recent log data from the journal (try with `cron.service`)
- `systemctl cat unit` ⇒ dumps the unit file
- `systemctl list-dependencies unit` ⇒ prints the dependence tree of a unit
- `systemctl show unit` ⇒ dumps properties of a unit

Several `systemctl` commands allow us to stop/start/enable/disable a unit immediately or on next boot:

- `systemctl stop unit` ⇒ immediately deactivates the *unit*
- `systemctl start unit` ⇒ immediately activates the *unit*
- `systemctl restart unit` ⇒ immediately stops and then starts the *unit*
- `systemctl enable unit` ⇒ the *unit* will be activated on next boot
- `systemctl disable unit` ⇒ the *unit* will be not be activated on next boot

Finally, we can modify the configuration of a unit by editing its unit file. The editor that will be employed can be selected by setting an environment variable (`SYSTEMD_EDITOR`, `EDITOR` or `VISUAL`); otherwise, `nano`, `vim` or `vi` are used (the first one that exists in the system).

- `systemctl edit unit` ⇒ opens for editing the unit file of the *unit*

Check the manual to discover more functionalities, filtering options, environment variables, location of unit files, ...

### 1.7.4 systemd ecosystem

`systemd` ecosystem includes several daemons and commands.

- `systemd-journald.service` and `journalctl` : daemon and command that queries the journal respectively. By default, `journalctl` command dumps the entire journal contents but it is possible filtering the output by unit, age, priority, ...
  - `journalctl --since "1 hr ago"` ⇒ checks messages from last hour
  - `journalctl -n N` ⇒ checks last *N* messages
  - `journalctl --disk-usage` ⇒ checks total journal disk usage
  - `journalctl -u unit` ⇒ checks message entries generated by *unit*
- `systemd-logind.service` : login manager
  - `journalctl -u systemd-logind.service`
- `systemd-networkd.service` : network manager, detects and configures network devices as they appear.
- `systemd-halt.service`, `systemd-poweroff.service`, `systemd-reboot.service`, `systemd-kexec.service`, `systemd-shutdown` : shutdown logic.

### 1.7.5 Advanced operations

`systemd` implements many other functionalities.

- `systemd --user` : ordinary users can create a `systemd` instance to manage its services and automated tasks, taking advantage of `systemd` features (socket activation, timers, journaling, ...)
- *masked* units: in addition to stopping and disabling units, units can be masked. A *masked* unit will remain inactive until unmasking it, no matter what else happens: start/enable the unit, OS reboot, ...
- `systemctl --host=hostname` : allows interacting remotely with a `systemd` instance
- `systemctl --state=help` : list all the possible states and substates per each unit type

# Bibliography

- [1] M. Kalle and M. Welsh, *Running Linux*, 5th ed. Sebastopol, USA: O'Reilly Media, 2005.
- [2] L. Wirzenius, J. Oja, S. Stafford, and A. Weeks, *The Linux System Administrators' Guide, version 0.9*. The Linux Documentation Project. TLDP. [Online]. Available: <http://www.tldp.org/LDP/sag/sag.pdf>
- [3] M. T. Jones, "Inside the linux boot process," IBM Developer Works, 2006. [Online]. Available: <http://www-128.ibm.com/developerworks/linux/library/l-linuxboot/?ca=dgr-Inxw06LinuxBoot>
- [4] "Drive naming in linux." [Online]. Available: <https://tldp.org/HOWTO/Partition-Mass-Storage-Definitions-Naming-HOWTO/x99.html>
- [5] S. Beateay, "Everyone Hates systemd," <https://betterprogramming.pub/why-most-linux-users-hate-systemd-c591eef3d034>, accessed: 2022-01-31.