

PARROT OS

ITP 51

OPERATING SYSTEM

TABLE OF CONTENTS

CHAPTER 01 INTRODUCTION	3
CHAPTER 02 PROCESS MANAGEMENT	5
CHAPTER 03 CPU SCHEDULING	8
CHAPTER 04 MEMORY MANAGEMENT	10
CHAPTER 05 STORAGE MANAGEMENT	13
CHAPTER 06 INPUT/OUTPUT SYSTEM	21
CHAPTER 07 FILE SYSTEM	25

CHAPTER 1

INTRODUCTION

Parrot OS Introduction:

Parrot OS, or commonly referred to as the Parrot Security OS, is a lightweight open-source computer OS originating at penetration testing, ethical hacking, and cybersecurity. It is a product of the Parrot Project community and came into existence in 2013. Parrot OS is forked out from Debian, one of the oldest and highly secured Linux distributions.

Parrot OS is designed specifically for human resources that are seeking security researchers, ethical hackers, and penetration testers that need a strong platform within which to test the security of computer systems. It comes with a large sort of pre-installed tools to do tasks as vulnerability assessments, digital forensics, reversing, and the environment of privacy. The operating system is light, fast, and secure made for both advanced users and newcomers to the cybersecurity world. It is also a Free and Open-source GNU/Linux distribution based on Debian Stable designed for security experts, developers and privacy aware people. It includes a full portable arsenal for IT security and digital forensics operations.

Considering the wide range of tools and features offered by Parrot OS, it can be a valuable alternative to Kali Linux for professionals in the cybersecurity field looking for a comprehensive and versatile toolkit. Considering the wide range of tools and features offered by Parrot OS, it can be a valuable alternative to Kali Linux for professionals in the cybersecurity field looking to expand their toolkit and explore different options for penetration testing and cybersecurity. By utilizing the capabilities of Parrot OS, cybersecurity professionals can enhance their penetration testing operations and gain access to a

diverse range of tools and features that complement those available in Kali Linux, further bolstering their expertise and effectiveness in securing networks and identifying vulnerabilities. Parrot OS is a popular and versatile operating system that offers a broad range of security tools and features for cybersecurity professionals. It provides a robust platform for penetration testing, vulnerability assessment, computer forensics, and anonymous web browsing. Parrot OS is a popular and versatile operating system that offers a broad range of security tools and features for cybersecurity professionals. It provides a robust platform for penetration testing, vulnerability assessment, computer forensics, and anonymous web browsing. Parrot OS is a valuable alternative to Kali Linux in the cybersecurity field. Over all, Parrot OS is a valuable alternative to Kali Linux in the cybersecurity field because of its wide range of tools and features, including options for customization and its focus on security. Parrot OS is a valuable alternative to Kali Linux for professionals in the cybersecurity field. It provides a comprehensive and versatile toolkit for penetration testing, vulnerability assessment, computer forensics, and anonymous web browsing.

Parrot OS History:

The development of Parrot OS has been a collaborative effort involving security experts, open-source developers, digital rights advocates, and Linux enthusiasts. Their passion and expertise have contributed to the evolution of Parrot OS, with regular updates and new releases over the years. This collaborative approach ensures that Parrot OS remains a cutting-edge and secure operating system for users who prioritize privacy and security. By incorporating the knowledge and skills of various contributors, Parrot OS has been able to address system issues, improve its functionality, and expand its compatibility with. Parrot OS has come a long way since its initial release in 2013. It has established itself as a reliable and powerful operating system for security professionals and enthusiasts. Over the years, it has undergone significant improvements in performance, stability, and user experience.

The introduction of Parrot OS 2.0 in 2015 marked a major milestone with its redesigned user interface and enhanced performance. This version solidified Parrot OS as a go-to distribution for penetration testing and security auditing purposes. In 2017, Parrot OS 3.0, codenamed “Lithium,” brought about substantial changes to the operating system. It featured an updated kernel, new tools, and improvements in system performance. The addition of the Parrot Anon Surf tool allowed users to browse the web anonymously, further enhancing privacy features. Two years later, Parrot OS 4.0, codenamed “Defon,” was released with a more refined interface, updated tools, and improved hardware support. This version continued to be popular among cybersecurity professionals due to its reliability and wide range of applications. In 2020, the Parrot OS team introduced a new versioning scheme aligned with Debian’s LTS releases to ensure better compatibility and stability. This change aimed to provide users with a more consistent experience by synchronizing with Debian’s long-term support cycle.

CHAPTER 2

PROCESS MANAGEMENT

Process management in Parrot OS, like in any Linux distribution, involves monitoring and controlling the execution of processes running on the system. Here are some key aspects of process management in Parrot OS.

To find process:

```
# pid [service_name]
```

Example:

```
# pidof sshd  
# pidof top  
# pidof httpd  
etc.
```

To start a process in background:

```
# firefox &
```

To check background process:

```
#jobs
```

To send a background process to foreground:

```
# fg [job id]
```

To check active Linux processes:

```
# ps -aux
```

For dynamic real time view of running system processes:

```
# top
```

To control process based on their names:

```
#pgrep    (pgrep looks through the currently running processes and lists the process IDs which match the selection criteria to stdout.)
```

To kill a process:

```
# pkill
```

There are multiple signals to send a process:

```
# kill -l //list signals  
  
# kill [signal id] [pid] (send a signal to process)
```

To kill an application process using its time:

```
# killall [program name]  
# pkill [program name]
```

Control and packages/programs:

To update packages:

```
# apt update
```

To upgrade packages:

```
# apt upgrade
```

To install packages:

```
# apt install packagename
```

To remove packages:

```
# apt remove packagename
```

To purge a package:

```
# apt purge packagename
```

To remove corrupted and unused packages run:

```
# apt autoremove // always run it after upgradation
```

By understanding and effectively using these commands and tools, system administrators, developers, and security professionals can effectively manage processes, identify resource-intensive processes, troubleshoot issues related to system performance, and ensure optimal utilization of system resources.

CHAPTER 3

CPU SCHEDULING

CPU scheduling is the process of managing the switching, between executing processes. It plays a role in systems ensuring that the computers CPU is utilized efficiently to maximize productivity. Two primary types of CPU scheduling exist; preemptive and non preemptive. Preemptive scheduling occurs

when a running process transitions to a state or when a waiting process becomes ready. On the hand nonpreemptive scheduling is used when a process completes or moves, from running to the waiting state.

ParrotOS, in common with other operating systems, employs a CPU scheduling algorithm to oversee process execution on the CPU. This CPU scheduler is responsible for determining the next process to run and allotting CPU time accordingly. The specifics of the CPU scheduling algorithm can vary based on the operating system version and its configuration.

As of my last update in January 2022, ParrotOS is a Debian-based Linux distribution tailored for penetration testing, ethical hacking, and security research. The default scheduler for Linux systems, including ParrotOS, is the Completely Fair Scheduler (CFS).

The Completely Fair Scheduler (CFS) is a process scheduler designed to provide fair distribution of the CPU time among processes. It uses a virtual timeline to track the amount of CPU time each process has received and tries to allocate CPU time in a fair and efficient manner.

First Come First Serve Scheduling Algorithm:

This is the first type of CPU scheduling algorithm. In this particular algorithm, we will learn how the CPU allocates resources to specific processes. In the First Come, First Serve CPU Scheduling Algorithm, the CPU allocates resources to processes in a sequential order. The allocation is done on a first-come-first-serve basis, where the resource goes to whichever process appears first. “First come, first served” can be shortened to FCFS.

Shortest Job First CPU Scheduling Algorithm:

This is another type of CPU scheduling algorithm. In this CPU scheduling algorithm, we will learn how the CPU allots resources to certain processes. The Shortest Job is heavily dependent on the Burst Times. Every CPU Scheduling Algorithm essentially relies on Arrival Times. In the case of this algorithm, known as Shortest Job First CPU Scheduling, resources are assigned by the CPU to processes in the ready queue with a minimum Burst Time. If we face a situation where two processes are present in the Ready Queue and their Burst Times are equal, either process can be chosen for execution. In real Operating Systems, if this problem occurs then resources will be allocated sequentially. Shortest Job First, commonly abbreviated as SJF, refers to a scheduling algorithm.

Priority CPU Scheduling:

In this particular CPU Scheduling Algorithm, we will explore how the CPU assigns resources to specific processes. Unlike other CPU scheduling algorithms, the Priority CPU Scheduling

assigns a specific priority number to each process. Two kinds of Priority Values exist: the value with the highest number is deemed to be of the utmost importance. And the value with the lowest number is regarded as having the lowest priority. Preventing Problems Takes Precedence When using a preemptive CPU Scheduling Algorithm, the order of operations is determined by priority. To ensure effective execution of each function in this method, it's important to prioritize the most critical steps first. In instances where multiple processors have equal value and there is potential for conflict, the FCFS (First Come First Serve) approach remains key to ensuring optimal utilization of CPU resources.

Round Robin CPU Scheduling:

The Round Robin CPU scheduling mechanism assigns each task a specific time slot by cycling through them. It is similar to First come, First served with preemptive mode and often uses the Time-Sharing method as well.

CHAPTER 4

MEMORY MANAGEMENT

The concept of “memory” refers to a collection of data organized in a format. It serves the purpose of storing instructions and processing information. This collection consists of words or bytes each with its location. The main goal of a computer system is to execute programs that need to be stored in the memory along, with their associated data while they are being executed. During operation the CPU retrieves instructions from memory based on the program counter value. Proper management is crucial for achieving use of memory and enabling multi programming. There are techniques for managing memory each, with its effectiveness depending on specific circumstances.

Memory management subsystem of Linux is a subsystem that manages the memory of the system. It has got the implementations such as demand paging and virtual memory. Moreover, it also includes memory allocation for both user space programs and kernel internal structures. For instance, in this case, we have the Linux memory management subsystem maps files into the address space of processes and other various functions; on another note, there is a lot of technical terminology in Linux memory management. This paper will explore some mechanisms used in Linux Memory Management as well as ways through which one can understand these mechanisms effectively.

Concept Overview:

The management of memory in Linux is a complex system that includes numerous functionalities to support various systems, ranging from MMU-less micro controllers to supercomputers. For computer systems, memory management without the Memory Management Unit (MMU) is called “nommu.” In the future, there should be a document written specifically for it. However, some similarities still exist between these concepts

Huge Pages:

Translating addresses requires multiple memory accesses which are much slower than the speed of the CPU. To avoid wasting valuable processor cycles on address translations, CPUs use a cache called Translation Lookaside Buffer (TLB) to handle such translations.

Virtual Memory Primer:

Physical memory in a computer system is one of the scarce resources. The physical memory need not be contiguous, but may in fact only exist as a series of ranges (or pages) of addresses. Besides, different CPU architectures and variations on the same architecture have their own ways of describing such ranges. Dealing directly with physical memory can be very hard. To simplify that complexity, a virtual memory mechanism was defined

Zones:

Linux groups memory pages together in different zones of usage. For example, ZONE_HIGHMEM will consist of memory not permanently mapped to the kernel’s address space. ZONE_DMA represents memory accessible by different devices for DMA, and ZONE_NORMAL refers to normally addressed pages.

Page Cache:

One common method for getting data into memory is to read it from files, because physical memory can be unreliable. Then the data will be placed in the page cache, so that during subsequent reads of any file no-costly disk access is required. Likewise, any file written to will have its data stored in the page cache and then bubbled out onto the backing storage device.

Nodes:

NUMA –Non-Uniform Memory Access systems can include multi-process or machines. In these types of systems, the memory is arranged into banks which have different access latency according to their distance from the processor. All nodes are identified as banks, and for all of them Linux provides a separate subsystem to manage memory. Each interface node has its own zone set, list of used and free pages as well as assorted statistical counters.

Anonymous Memory:

This literally means memory without a file system. These mappings are usually made for the heap and stack areas of memory used by a particular program, either implicitly or through explicit calls to the system call. In fact, anonymous mappings simply define virtual memory areas that a given program can access.

OOM Killer:

There is a high probability that the memory reclamation of system kernel will be insufficient, releasing too late for you to pass on instructions; run out of machine's memory and leave your task unfinished.

Compaction:

When the system is running, various tasks allocate and DE-allocate memory space which becomes partitioned in this way. But scattered physical pages can be virtualized by memory. Memory compaction addresses partitioning issues.

Reclaim:

How the page is used determines how Linux memory management treats it. There are also pages that either do not cache details which already exist elsewhere on a hard disk or because they could be swapped in again to the process this is called reclaimable.

CHAPTER 5

STORAGE MANAGEMENT

Storage management encompasses the procedures and technologies that firms use to preserve their data storage system. The purpose of storage management is to help organizations achieve a balance between cost, performance and capacity. It also works to protect sensitive information, while at the same time making it available throughout an organization.

Every file in the GNU/Linux operating system has both a user and group attached to it. The person who created a file is its owner, and their main existing group will be that of the said object. As an example of this second concept; throughout previous chapters we worked under the name used. If that account created new files, they would be linked to parrot user status as well as belonging to default groups built into its infrastructure. Running sudo commands is often necessary when executing critical operations such as reading or editing programs/ files and changing access permissions on any of them.

Let's analyze the output of the command `ls -l` :

```
[root@parrot]-[/home/parrot]
# ls -l archive.txt
-rw-rw-r-- 1 parrot hackers 0 oct 16 12:32 archive.txt
drwxr-xr-x 3 parrot hackers 4096 oct 15 16:25 scripts
```

When using the command `ls -l`, information regarding whether a file is (-) or directory (d), its permissions in terms of `rw-rw-r--`, as well as other fields such as user and group ownership (e.g. `parrot hackers`), size, last modification date (Oct 16 12: 32) and name are displayed. To get an initial grasp of this output, let us look at the first few fields that set permissions for files and folders owned by users or groups. There is a simple three-tiered scheme for regulating access permissions of these items on Linux systems.

Read permission, represented by the “r” letter.

Write permission, represented by the “w” letter.

Execution permission, represented by the “x” letter.

In the case of a .txt file, it has the following permissions:

Owner	Group	Other	Users
r	w	-	r

This implies that the file is readable by all system users, but only modifications can be made to it by either the owner or members of their respective group.

To calculate the value, we will base the sum of its decimal values according to the following correspondence:

Permission	r	w	x
Decimal Value	4	2	1

That is, the decimal value for the read permission is 4, the value for write permission is 2 and the value for execute permission is 1. The possible values are as follow:

Permission	Value
rwX	7
rw-	6
r-X	5
r--	4
-wX	3
-w-	2
--r	1
---	0

Therefore, we come to the following conclusion:

Permission	Value
<code>rwX rwX rwX</code>	<code>777</code>
<code>rwX r-X r--</code>	<code>754</code>
<code>r-X r-- ---</code>	<code>540</code>

Chmod – Helps to manage the files and folders permissions.

Basic syntax of chmod:

```
$ chmod [mode] [permissions] [file or directory]
```

In the example below, we have a script folder in which not all scripts have the execute permission.

```
└─[root@parrot]─[/home/parrot]
```

```
└─ #ls -l scripts/
```

```
total 16
```

```
-rw-r--r-- 1 parrot hackers 932 oct 18 01:06 ddos-detect.py
```

```
-rwxr-xr-x 1 parrot hackers 235 oct 18 01:06 ping.sh
```

```
-rwxr-xr-x 1 parrot hackers 780 oct 18 01:17 wireless-dos-ids.py
```

```
-rw-r--r-- 1 parrot hackers 1587 oct 18 01:05 wireless-dos.py
```


As you can see in the execution of `ls -l scripts/`, some scripts have execution permissions for all the system users (which is not recommended), while others do not have execution permission even for the owner user. To correct this error we apply the following:

```
└─[root@parrot]─[/home/parrot]
```

```
└─ #chmod -R 770 scripts/
```

```
└─[root@parrot]─[/home/parrot]
```

```
└─ #ls -l scripts/
```

```
total 16
```

```
-rwxrwx--- 1 parrot hackers 932 oct 18 01:06 ddos-detect.py
```

```
-rwxrwx--- 1 parrot hackers 235 oct 18 01:06 ping.sh
```

```
-rwxrwx--- 1 parrot hackers 780 oct 18 01:17 wireless-dos-ids.py
```

```
-rwxrwx--- 1 parrot hackers 1587 oct 18 01:05 wireless-dos.py
```

Another way to add or remove permissions is using these modes:

a → indicates that it will be applied to all

u → indicates that it will be applied to the user

g → indicates that it will be applied to the group

o → indicates that it will apply to others

+ → indicates that the permission is added

- → indicates that the permission is removed

r → indicates read permission

w → indicates write permission

x → indicates execution permission

The basic syntax for using “chmod” with these modes is as follows:

```
chmod [a | u | g | o] [+ | -] [r | w | x]
```

Possible combinations:

- a+r Read permissions for all
- +r As before, if nothing is indicated, ‘a’ is assumed.
- og-x Removes execution permission from all but the user.
- u+rwX Gives all the permissions to the user.
- o-rwx Remove the permissions from the others

Chown – With the system utility chown (change owner), you can change file ownership.

Its basic syntax is as follows:

```
$ chown [options] [owner]: [group (optional)] [files or directories]
```

Chown options:

- -R → Recursively changes the owner of the directories along with all its contents.
- -v or –verbose → Used to show a more descriptive output.
- --version → See the version number of the program.
- -dereference → Acts on symbolic links instead of on the destination.
- -h or –no-deference → In the case of symbolic links, change the owner of the destination instead of the link itself.
- --reference → Changes the owner of a file, taking as reference the owner of the other.

Example of use:

```
└─[root@parrot]─[/home/parrot]
```

```
└─ #ls -l scripts/
```

total 16

```
-rwxrw---- 1 parrot parrot 932 oct 18 01:06 ddos-detect.py
```

```
-rwxrw---- 1 parrot parrot 235 oct 18 01:06 ping.sh
```

```
-rwxrw---- 1 parrot parrot 780 oct 18 01:17 wireless-dos-ids.py
```

```
-rwxrw---- 1 parrot parrot 1587 oct 18 01:05 wireless-dos.py
```

```
└─[root@parrot]─[/home/parrot]
```

```
└─ #chown -R root:root scripts/
```

```
└─[root@parrot]─[/home/parrot]
```

```
└─ #ls -l scripts/
```

total 16

```
-rwxrw---- 1 root root 932 oct 18 01:06 ddos-detect.py
```

```
-rwxrw---- 1 root root 235 oct 18 01:06 ping.sh
```

```
-rwxrw---- 1 root root 780 oct 18 01:17 wireless-dos-ids.py
```

```
-rwxrw---- 1 root root 1587 oct 18 01:05 wireless-dos.py
```

Chgrp – The chgrp command is used to change the group to which a file or directory belongs.

Basic syntax:

```
$ chgrp [options] [file (s)] or [directory (s)]
```

Options:

- -R -> Recursively changes the group to which the directories belong together with all their contents.
- -v (or --verbose) -> Used to show a more descriptive output.
- --version -> See the version number of the program.
- --dereference -> Acts on symbolic links instead of on the destination.
- -h (or --no-dereference) -> In the case of symbolic links, change the destination group instead of the link itself.
- --reference -> Change the group of a file taking as reference the owner of another.

Example of chgrp:

```
└─[root@parrot]─[/home/parrot]
```

```
└─ #ls -l scripts/
```

total 16

```
-rwxrw---- 1 parrot parrot 932 oct 18 01:06 ddos-detect.py
```

```
-rwxrw---- 1 parrot parrot 235 oct 18 01:06 ping.sh
```

```
-rwxrw---- 1 parrot parrot 780 oct 18 01:17 wireless-dos-ids.py
```

```
-rwxrw---- 1 parrot parrot 1587 oct 18 01:05 wireless-dos.py
```

```
└─[root@parrot]─[/home/parrot]
```

```
└─ #chown -R root:root scripts/
```

```
└─[root@parrot]─[/home/parrot]
```

```
└─ #ls -l scripts/
```

```
total 16
```

```
-rwxrw---- 1 root root 932 oct 18 01:06 ddos-detect.py
```

```
-rwxrw---- 1 root root 235 oct 18 01:06 ping.sh
```

```
-rwxrw---- 1 root root 780 oct 18 01:17 wireless-dos-ids.py
```

```
-rwxrw---- 1 root root 1587 oct 18 01:05 wireless-dos.py
```

CHAPTER 6

INPUT/OUTPUT SYSTEM

In computer architecture, I/O operations are key to the sharing of data among separate machines. This includes everything from moving audio files, software instruction sets through text and video streams within a system. I/O functionality underpins essentially every kind of data movement.

Data transfer begins when the central processing unit (CPU) of a computer is sent instructions through input/output signals. These signals may come from the hardware, software or man. I/O signals are data channels between a digital CPU, storage controller or memory and the storage device. In contrast, output/input signals carry information from the computer to an output device. Some I/O devices can only pick up data, not send it back. On the other hand, output-only devices can neither send nor receive data and are only capable of accepting inputs from other machines. Furthermore, some I/O equipment is suitable not only for receiving input but also processing the variables and producing an appropriately responsive output.

How does a program know its inputs and outputs:

Let us begin with a brief overview of the fundamentals of operating systems. When a file is open, an index number is stored by the operating system that can be utilized to access and read it. This index in Unix based systems (including all Linux distributions) is known as a file descriptor whereas Windows refers to it as a handle, though both serve the same purpose. For simplicity's sake, this article will use Unix terminology and refer to these indices as file descriptors. When a file is opened in the program, the operating system provides you with a file descriptor to utilize for accessing it.

I have compiled a simple program called parrot, which receives input and prints the same output all the time until it receives the line "end":

```
pmihaylov@winterfell:~/personal-projects/playground$ ./parrot
hello
Parrot says: hello
there
Parrot says: there
█
```

So, I will let it run and open another terminal in the meantime. In it, I will list some processes and find my program.

```
pmihaylov@winterfell:~/personal-projects/playground$ ps -a
  PID TTY          TIME CMD
 23764 pts/19    00:00:00 parrot
 23802 pts/21    00:00:00 ps
```

Now, I will go to this magic Linux directory – /proc. It contains all the data about currently running processes in files (Everything in Linux is a file, including processes, but that's another topic). If I open the directory 23764 (the process id!), I will see the data of my parrot process. I open the directory fd, which holds info about all the file descriptors of that process and show you its contents:

```
pmihaylov@winterfell:/proc/23764/fd$ ls -l
total 0
lrwx----- 1 pmihaylov pmihaylov 64 юли 21 16:24 0 -> /dev/pts/19
lrwx----- 1 pmihaylov pmihaylov 64 юли 21 16:24 1 -> /dev/pts/19
lrwx----- 1 pmihaylov pmihaylov 64 юли 21 16:22 2 -> /dev/pts/19
```

Manipulating the IO:

Turns out, there are simple routines you can use in the terminal with which you can easily change the Standard IO. One is using the '>' command. So, I am in the parrot program's folder and this time I will run it using the following command.

```
./parrot > output.txt
```

This time, when I write messages, I don't see any output on the screen. All you see is the input I type in:

```
pmihaylov@winterfell:~/personal-projects/playground$ ./parrot > output.txt
hello
there
end
```

But if I open the file output.txt, you can see that all the output I expected is written in it:

```
output.txt (~/.personal-projects/playground) - gedit
1 Parrot says: hello
2 Parrot says: there
3 Parrot says: end
```

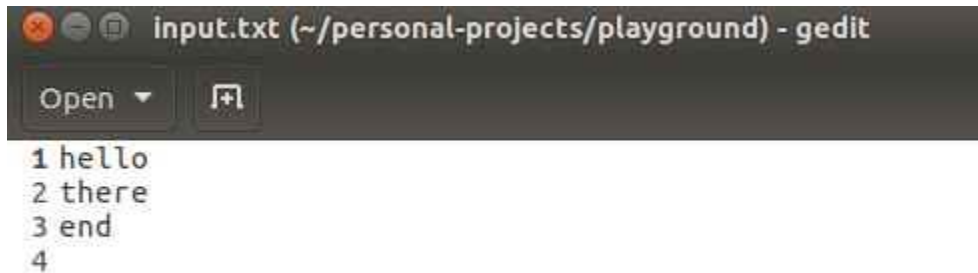
Now, doing all the routines I described in the demo from the previous section, I go to the proc directory and check out the file descriptors of my process. It turns out that file descriptor 1 now points to the file I just mentioned:

```
pmihaylov@winterfell:/proc/24122/fd$ ls -g
total 0
lrwx----- 1 pmihaylov 64 юли 21 16:31 0 -> /dev/pts/19
l-wx----- 1 pmihaylov 64 юли 21 16:31 1 -> /home/pmihaylov/personal-projects/playground/output.txt
lrwx----- 1 pmihaylov 64 юли 21 16:31 2 -> /dev/pts/19
```

Similarly, there is a '<' command which accepts the standard input from a file.

I create a file called input.txt and add some content. Next I execute:

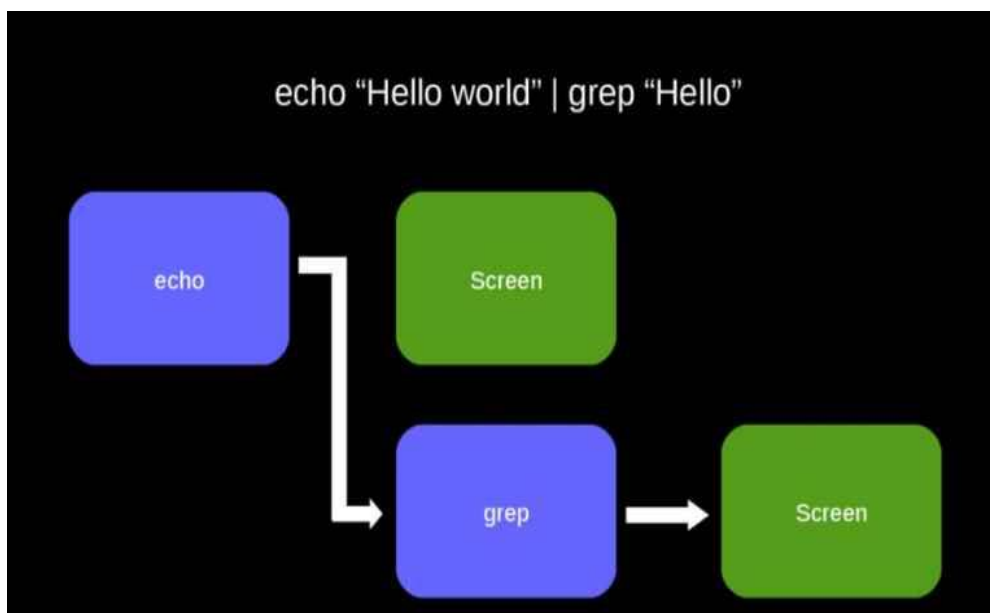
```
./parrot < input.txt
```

A screenshot of a terminal window titled "input.txt (~/.personal-projects/playground) - gedit". The window shows a text editor with four lines of text: "1 hello", "2 there", "3 end", and "4". The text is displayed in a monospaced font with line numbers on the left.

```
1 hello
2 there
3 end
4
```

What about processes?

It is also interesting to study communication methods among two processes by the same means. Pipes are a good tool, represented with the symbol '|', to do connect in order that what comes out of one process A can be put into another inputs B. In other words, information which is output from one procedure becomes input data for use by another procedure using pipes.



ls -l: lists the files from a given directory in rows

sort – sorts all input rows in alphabetical order

tail -n 3: takes the first three entries from all input rows

Using pipes, you can combine these together and you get:

```
ls -l | sort | head -n 3
```

The result is:

“ls -l” outputs all the files in rows. “sort” sorts them in alphabetical order and “head -n 3” takes only the first three rows:



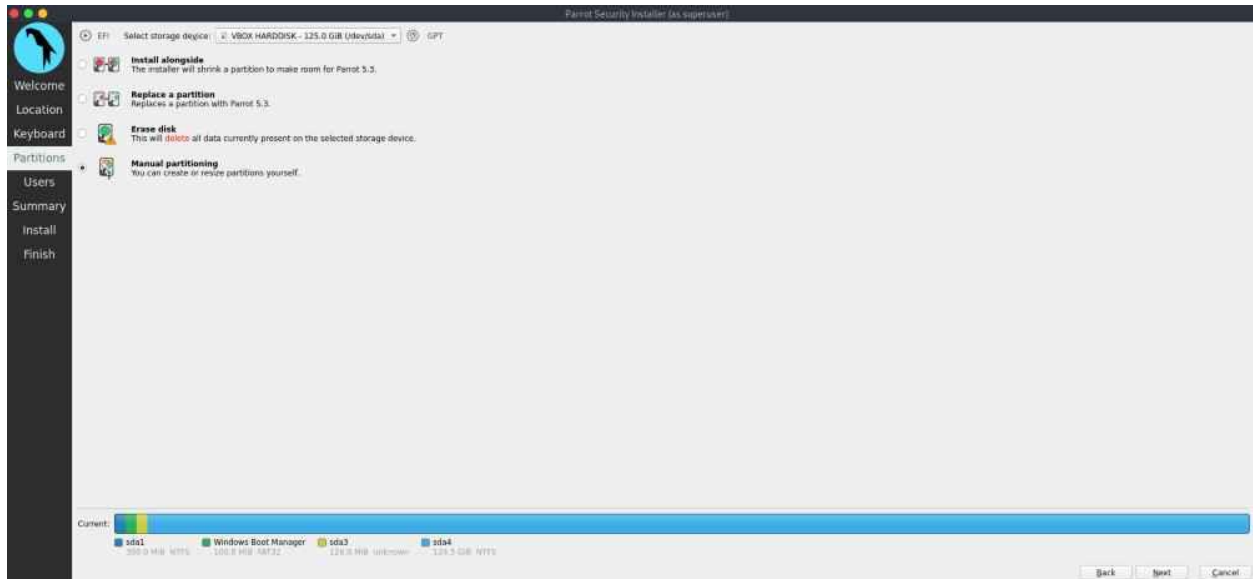
```
pmihaylov@winterfell:~/personal-projects/playground/test$ ls -l | sort | head -n 3
file1
file10
file2
```

CHAPTER 7

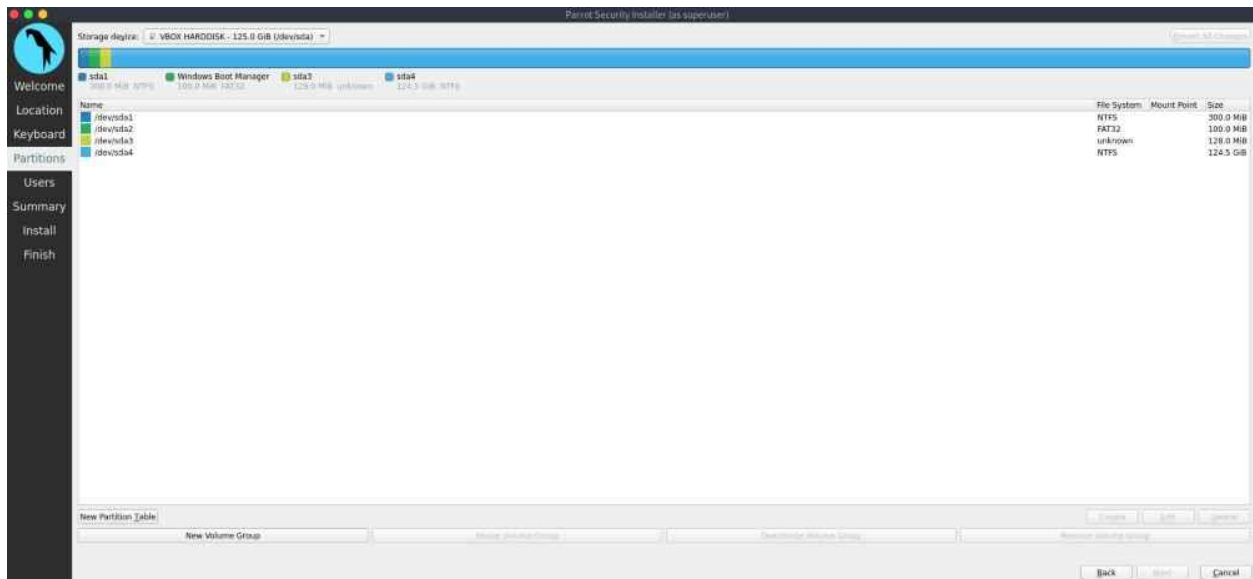
FILE SYSTEM

The word “filesystem” refers to how we give names and arrange files neatly for keeping and getting them in a computer. If we didn’t have this system, the information saved would be messy and hard to find. As technology grows and can store more data, it’s very important to manage files well for easy access. The names and arrangement of computer file systems and documents follow the rules of old-fashioned paper filing systems. They use smart ways to keep files organized for easy finding and getting them when needed. Differences in file systems can happen among various operating systems (OS) like Microsoft Windows, macOS and Linux-based ones. Some file systems are made for specific tasks. There are three main types of file system: distributed, disk-based and special purpose filesystems.

After following the steps for setting the Parrot Installation before partitioning, select Manual Partitioning then click on Next.



You'll see something similar to this:



The partitions in detail:

/dev/sda1 is a hidden partition which contains Windows Files for Recovery.

/dev/sda2 is the boot partition.

/dev/sda3 is MSR (Microsoft Reserved partition).

/dev/sda4 is where preinstalled OS exists (Windows 10 for this use case).

To make ParrotOS work in a UEFI computer, at least three working partitions are needed:

/boot/EFI – the folder containing the efi firmware necessary to boot the system.

/ - the folder containing the entire system

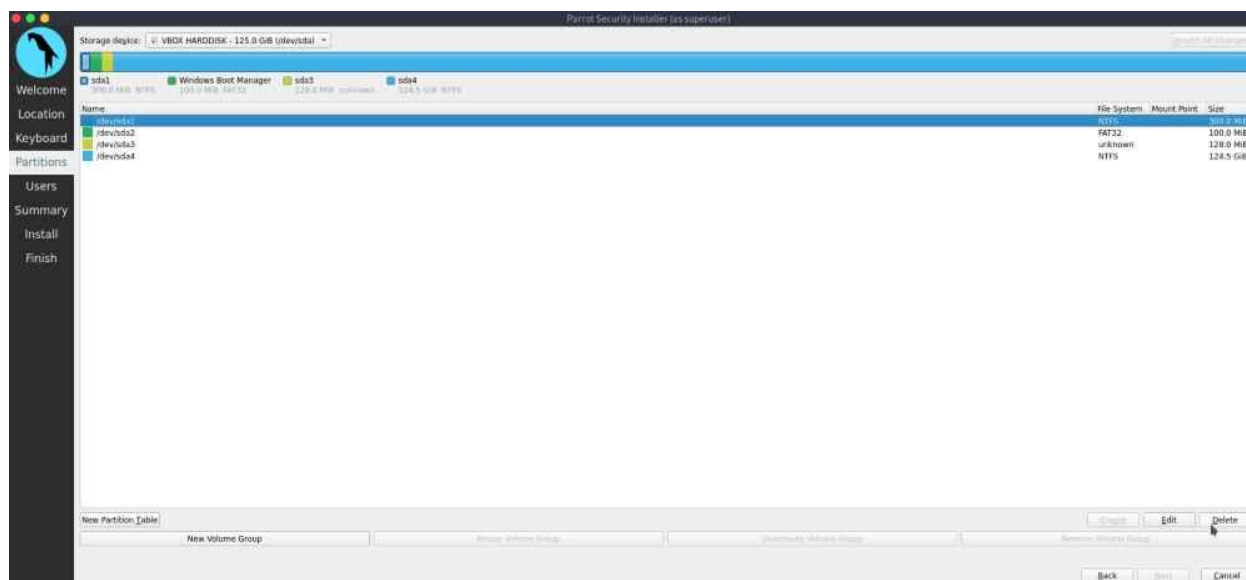
/home- the User data folder

In a standard BIOS partition, at least two working partitions are needed:

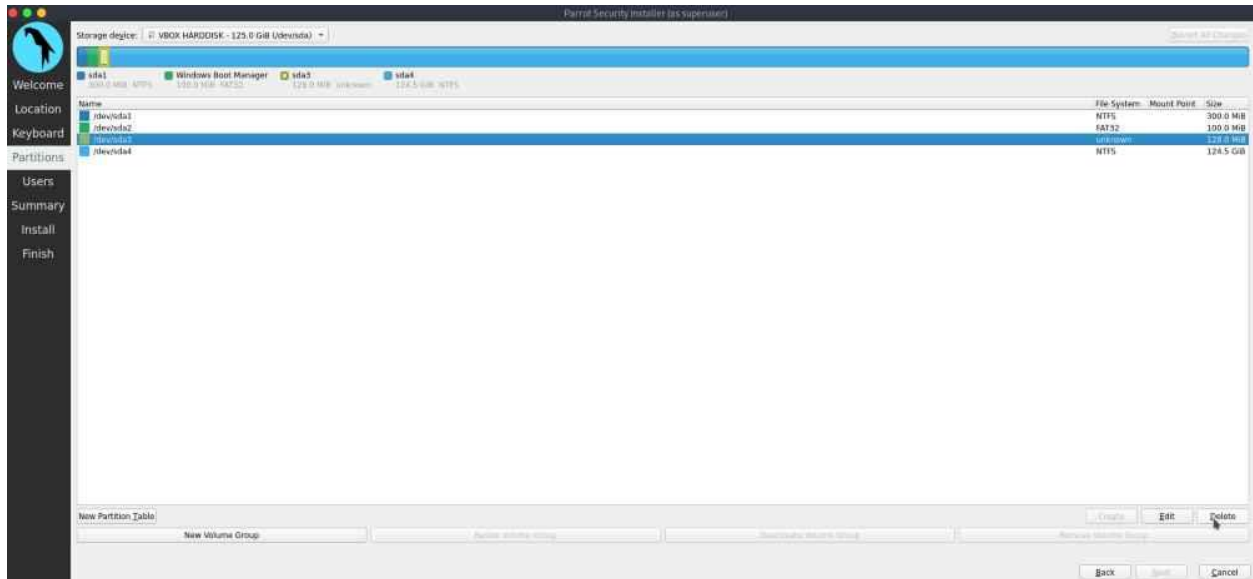
- /
- /home

/dev/sda1 and /dev/sda3 are useless for ParrotOS, so let's delete them.

Select /dev/sda1 then click on Delete:



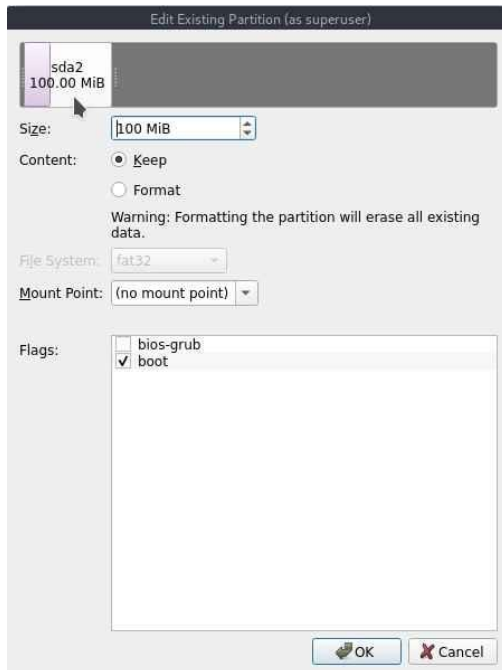
Repeat the operation for /dev/sda3. The configuration at this stage will be like this:



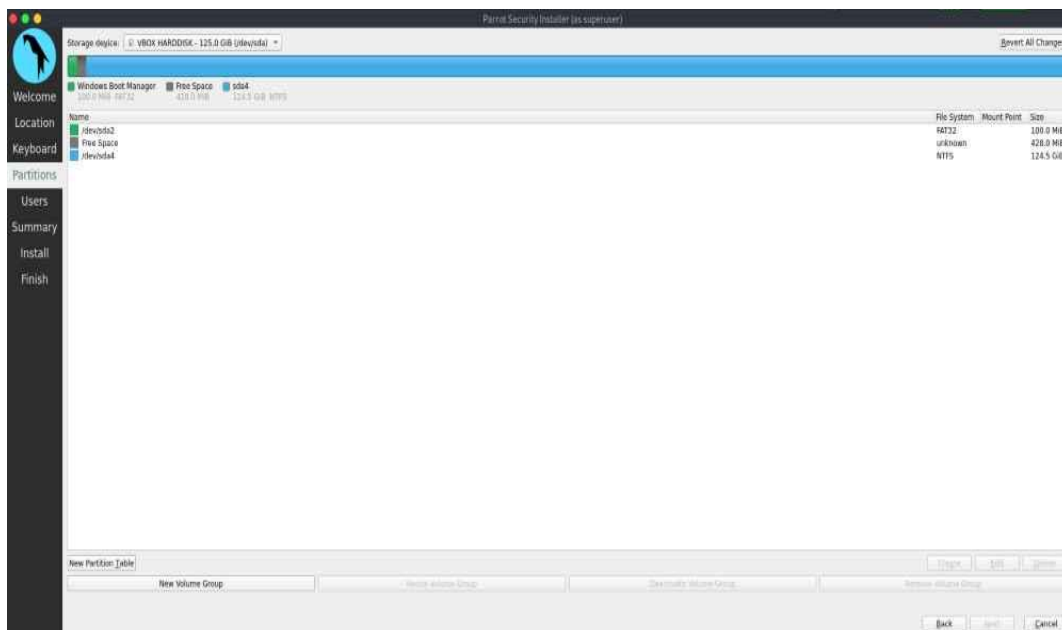
You may need to move /dev/sda2 to the first section of the disk, so click on Edit and this window will appear:



Hold and drag left the partition, then click on Ok:

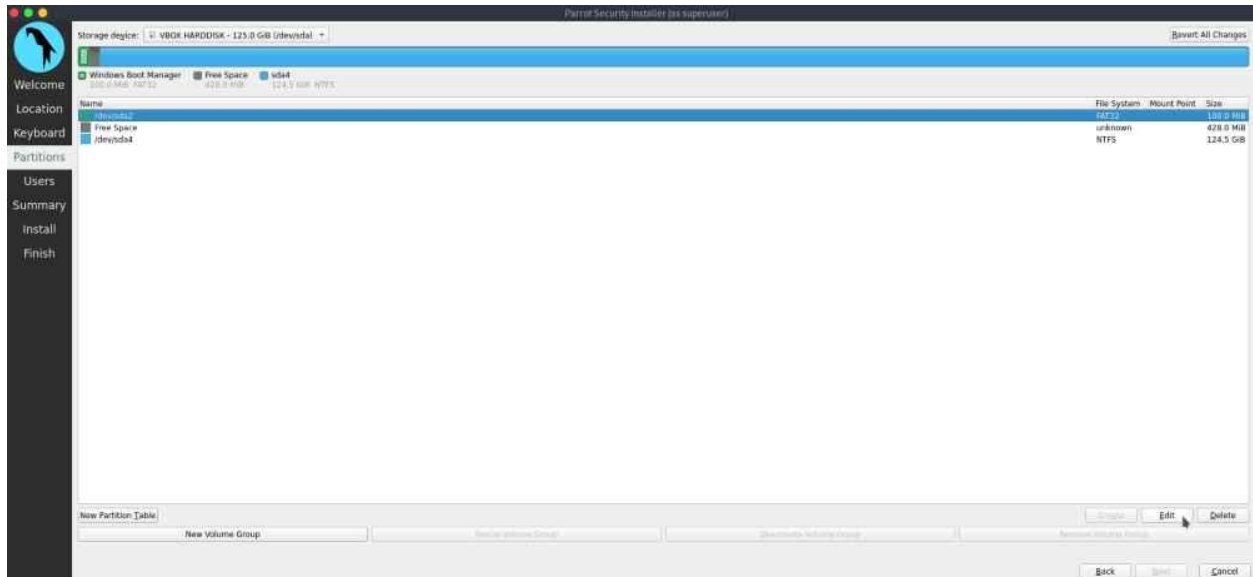


The configuration at this stage will be like this:

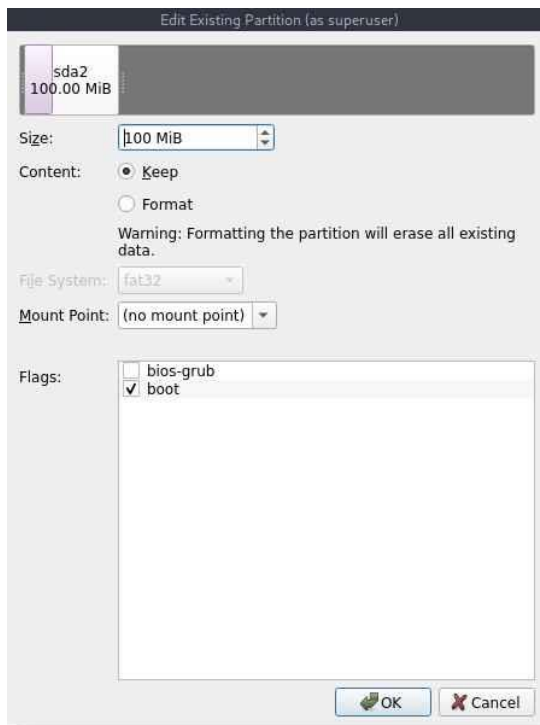


Now, let's change the mount point for the necessary partitions.

Select /dev/sda2 then click on Edit:

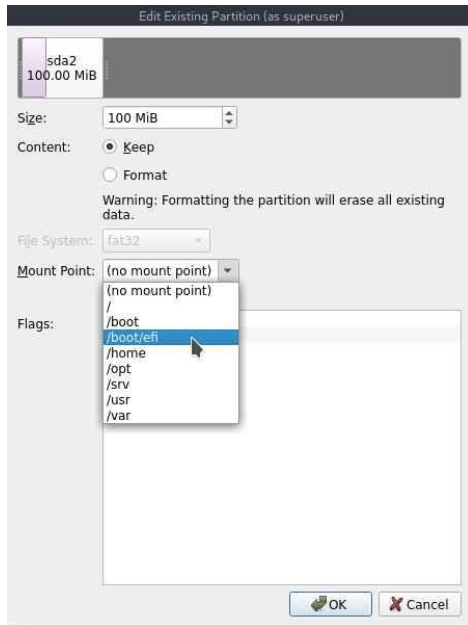


This window will open up:

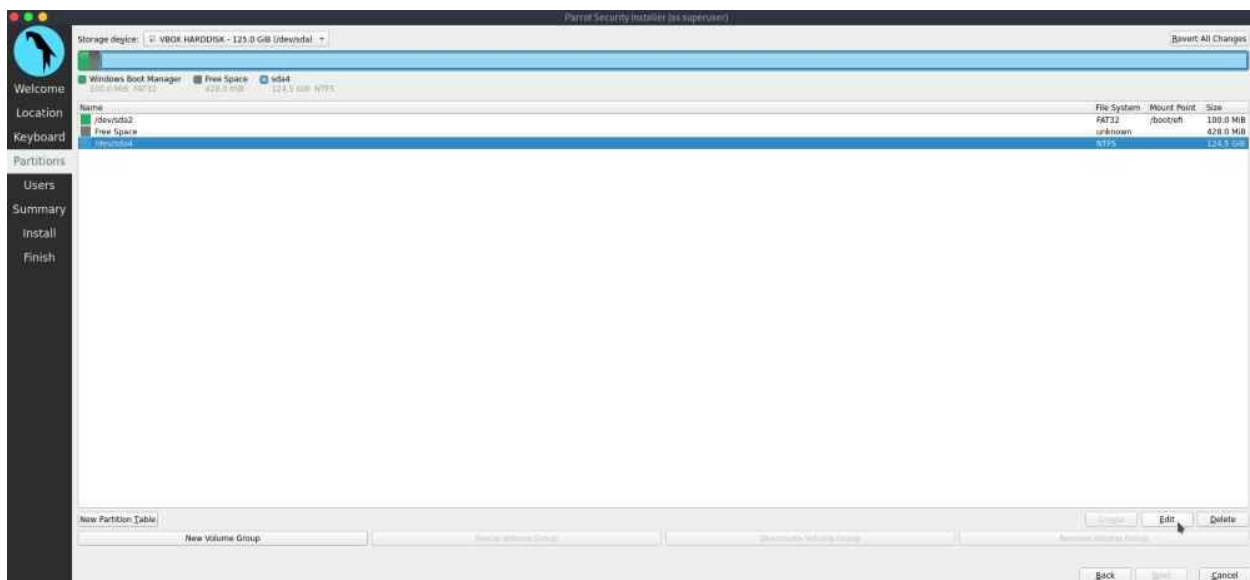


Here is possible to shrink/resize partitions (by dragging the bar or inserting the size in MiB), set flags and mount point.

Click on the Mount Point drop-down list, and set it on /boot/EFI, then click on OK:

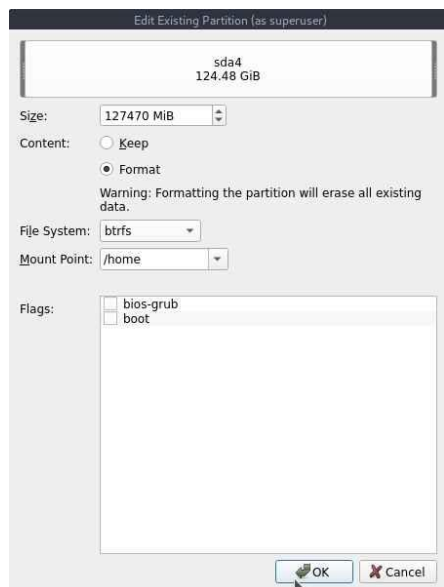


Select `/dev/sda4` then click on Edit:

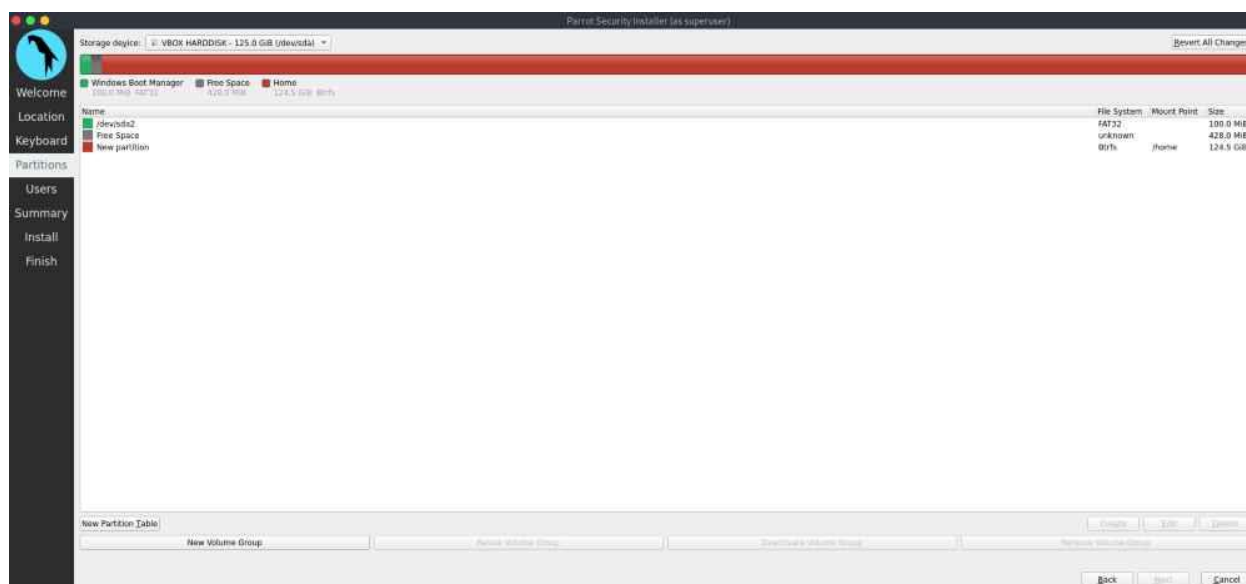


As mentioned before, at least three working partitions are needed for ParrotOS.

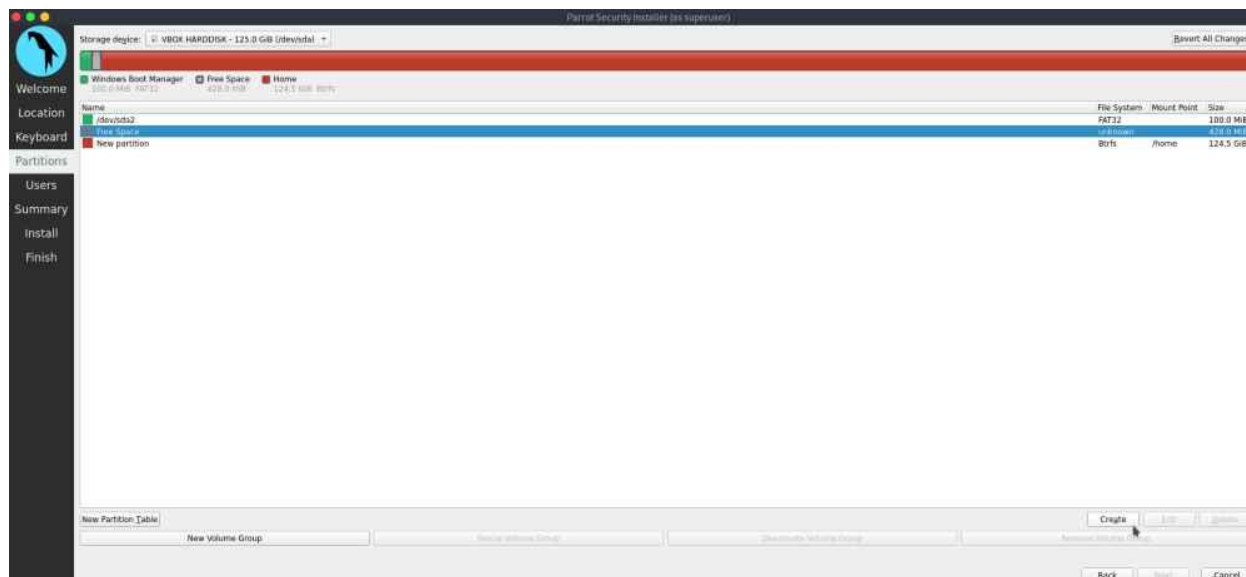
The first partition going to be created will be `/home` – the User data folder Shrink the partition by dragging right the slider, unless you'll have 20gb of unallocated space, then set the mount point to `/home`, set the content to Format, then click on Ok:



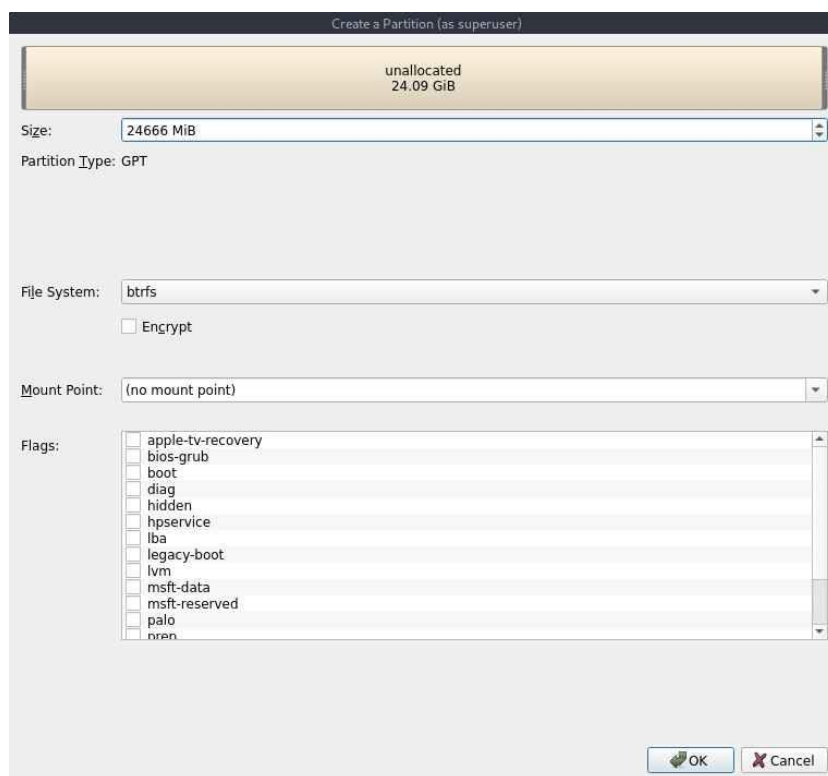
Now the updated situation will be this:



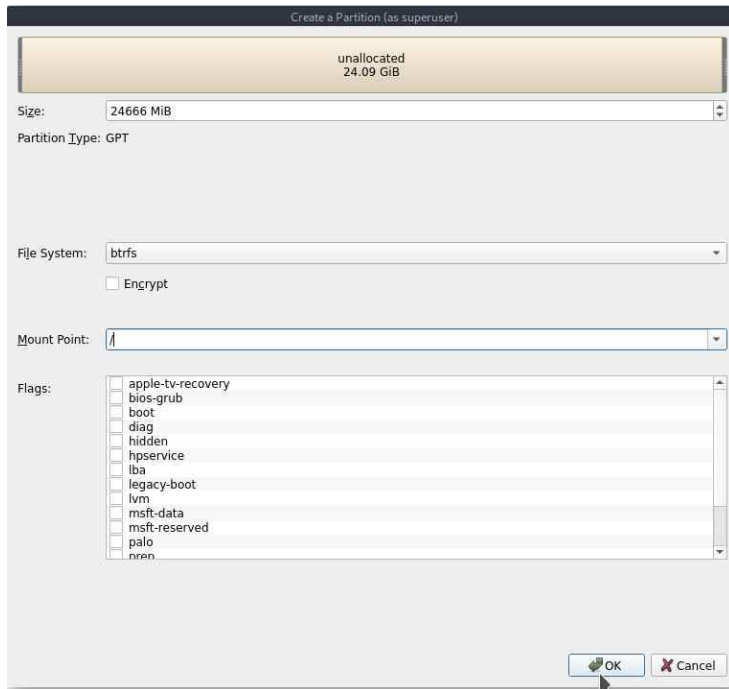
Select the Free Space and click on Create to setup / - the folder containing the entire system:



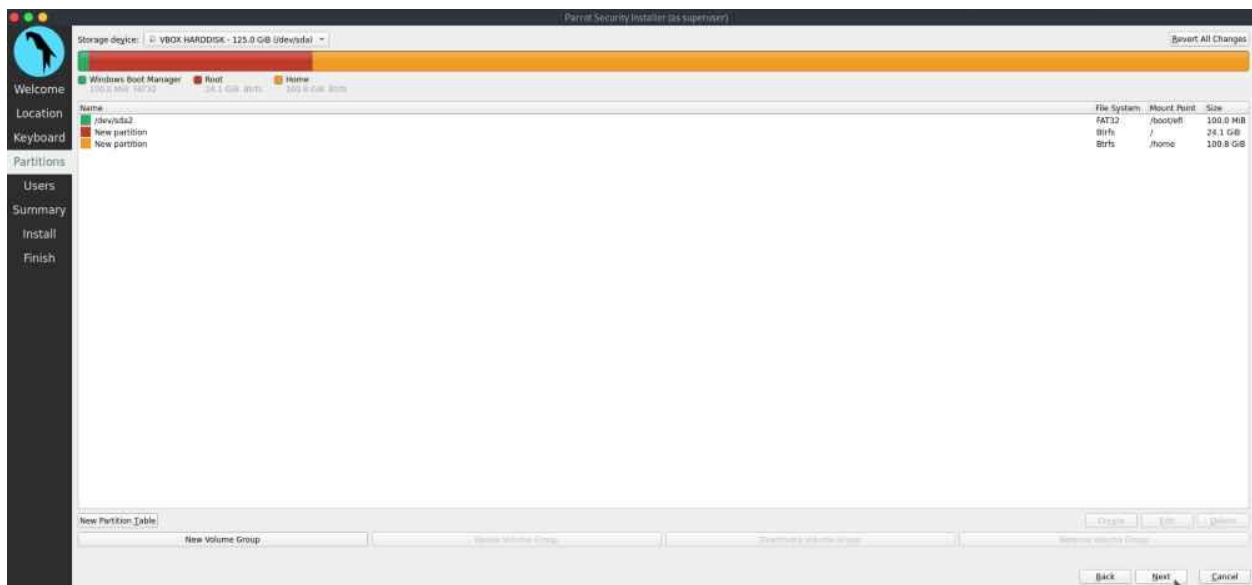
This window will appear:



Click on the Mount Point drop-down list, set the file system you want (ParrotOS uses BTRFS by default), set the mount point in / (root), then click on OK:

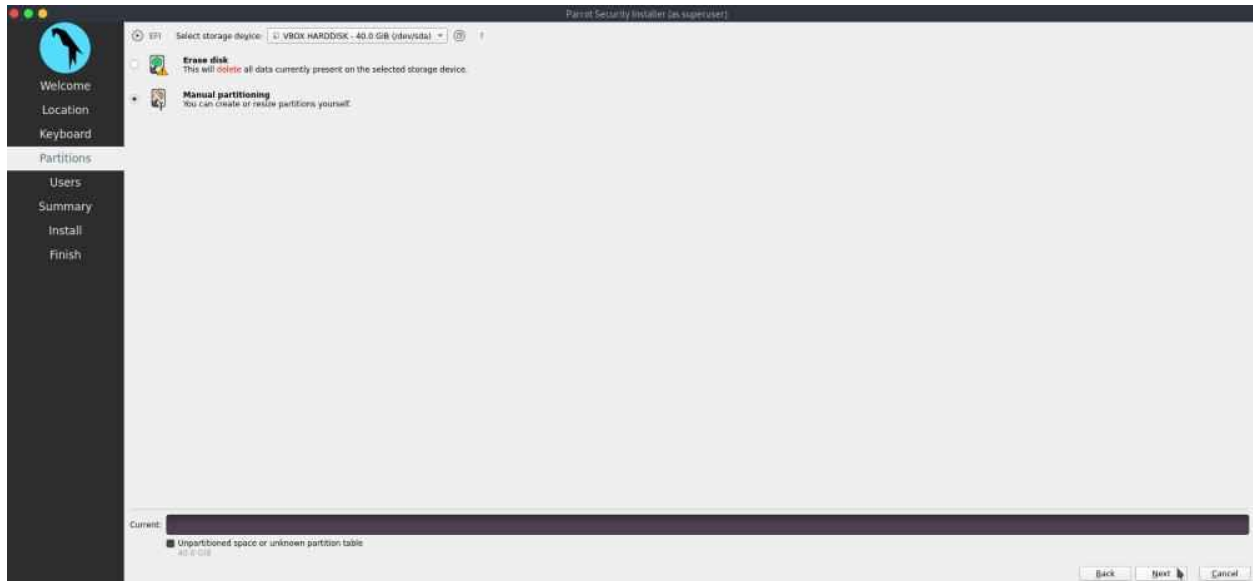


After this, proceed with the final steps of the installation by clicking on Next:

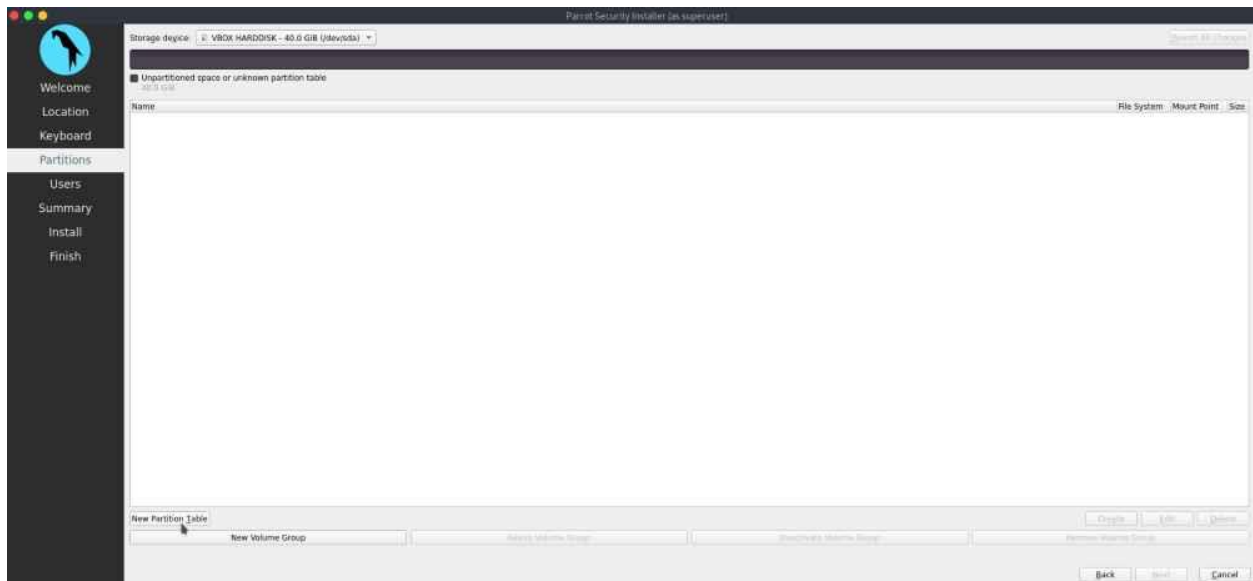


Case 2: Partitioning an empty disk

After following the steps for setting the Parrot Installation before partitioning, select Manual Partitioning then click on Next:



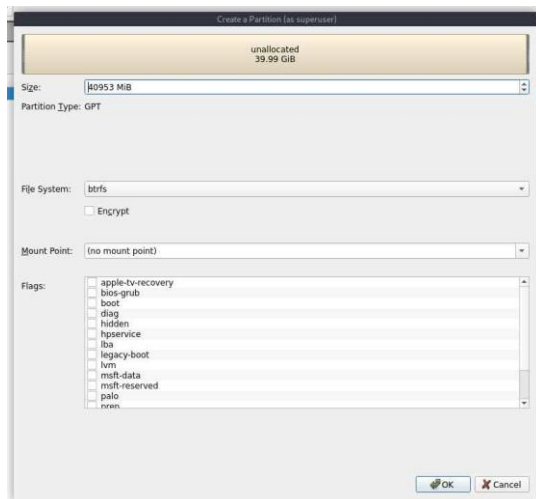
Since the hard drive is empty, the space will appear unallocated. Let's create a new partition table by clicking on New Partition Table:



A dialogue window will appear asking the desired partition table type, keep the default value (GPT) and click Ok:

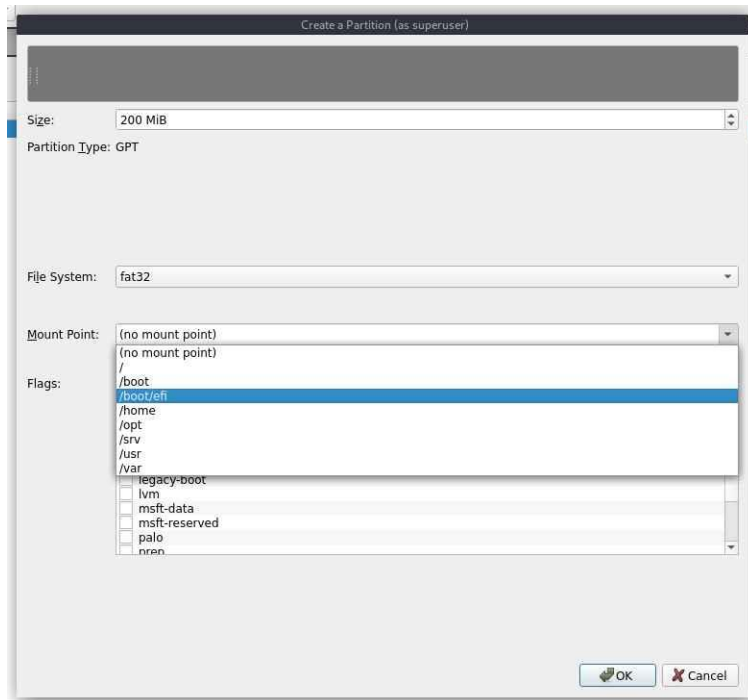


Select the Free Space and click on Create:

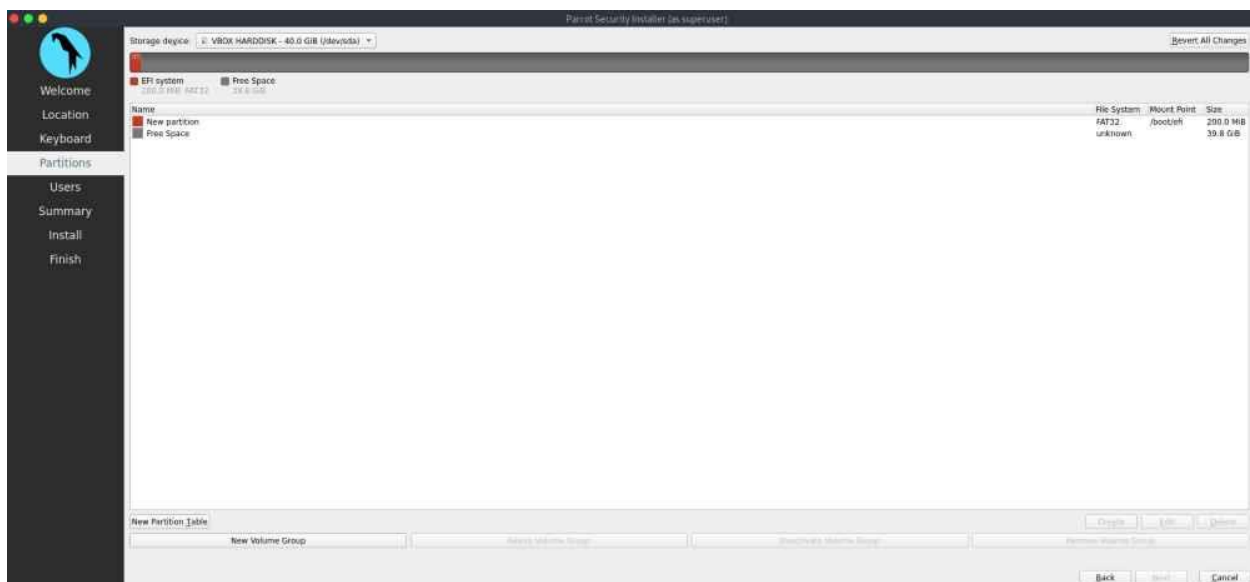


From here, it's possible to create and edit partitions. Let's create the first one, /boot/EFI – the folder containing the efi firmware necessary to boot the system. By following three simple steps:

- Click on the Mount Point drop-down list, and set it on /boot/EFI
- Click on File System drop-down list, set it on fat32
- On the Size text field, write 200MiB, then click on

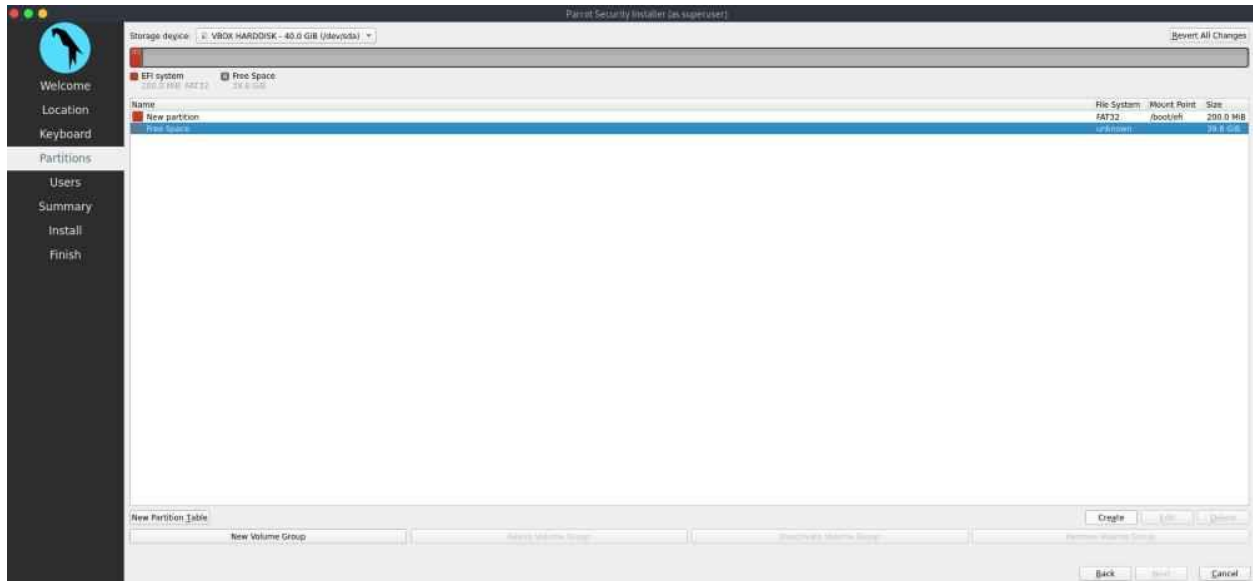


At this stage, the partition table will result like this:

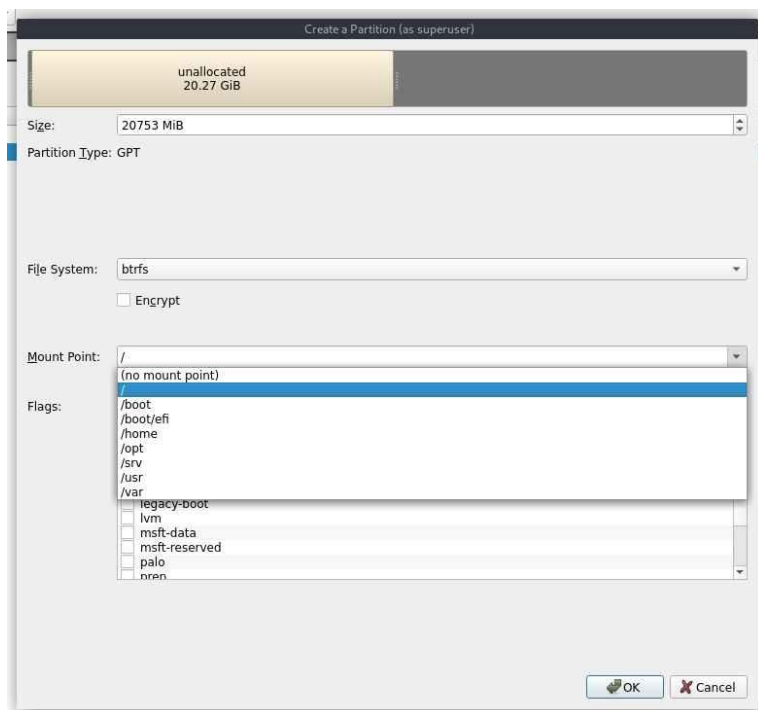


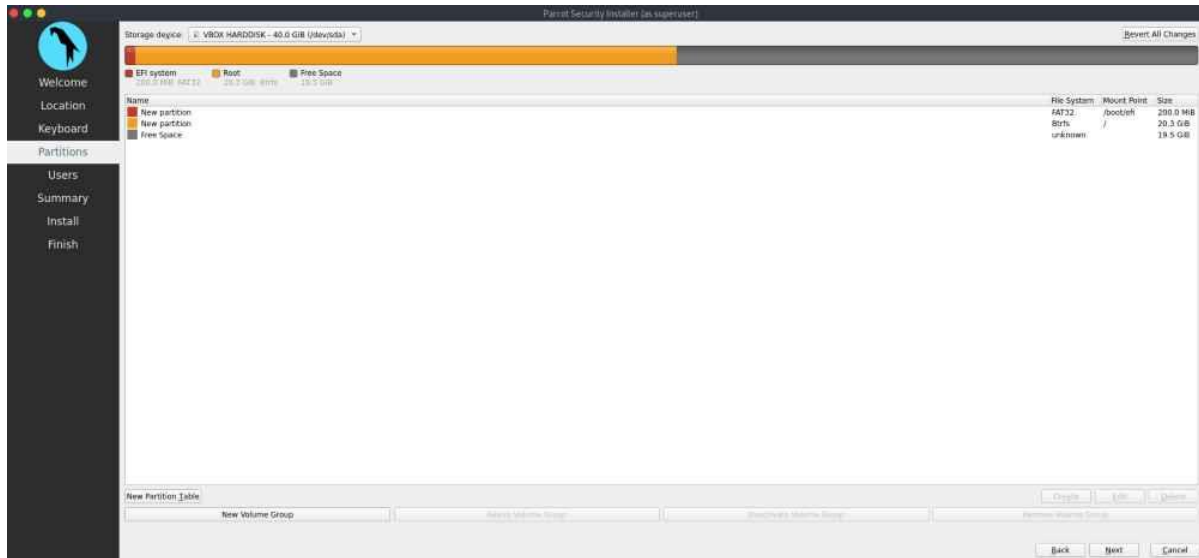
Now select again the Free Space and click on Create, let's create * /

- the folder containing the entire system



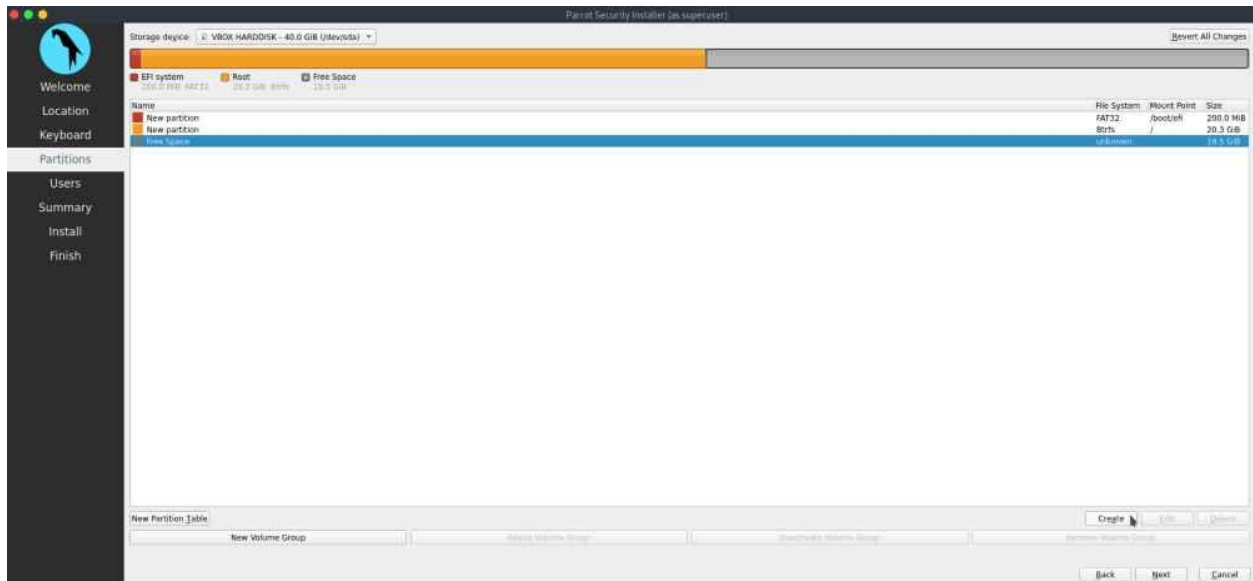
From the partition setup window, set the partition Size to 20753MiB, the File System to btrfs and the Mount Point to /, then click on Ok:



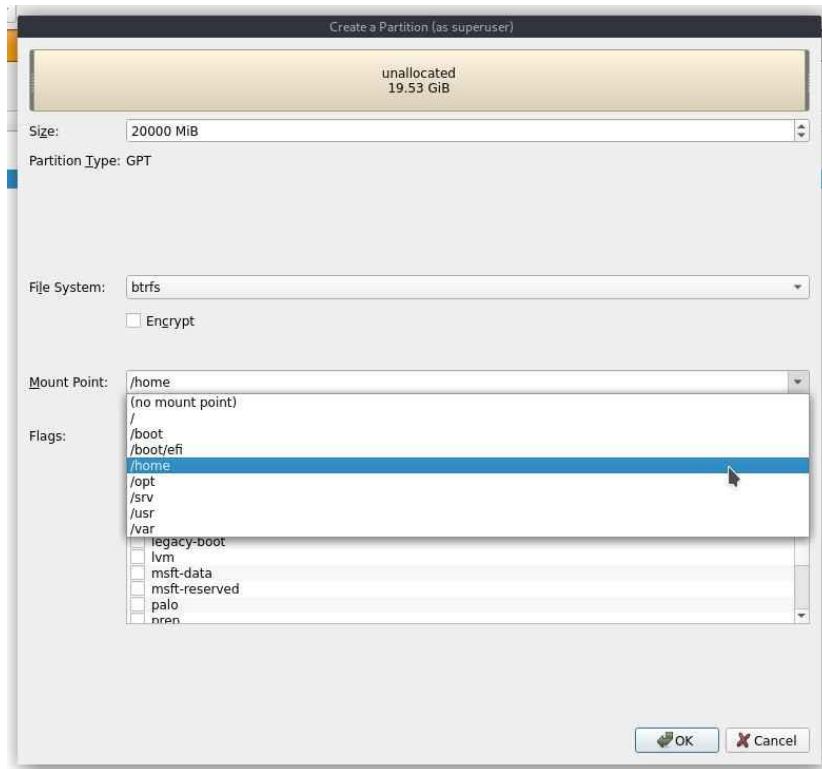


Finally, let's create /home- the User data folder with the remaining Free Space.

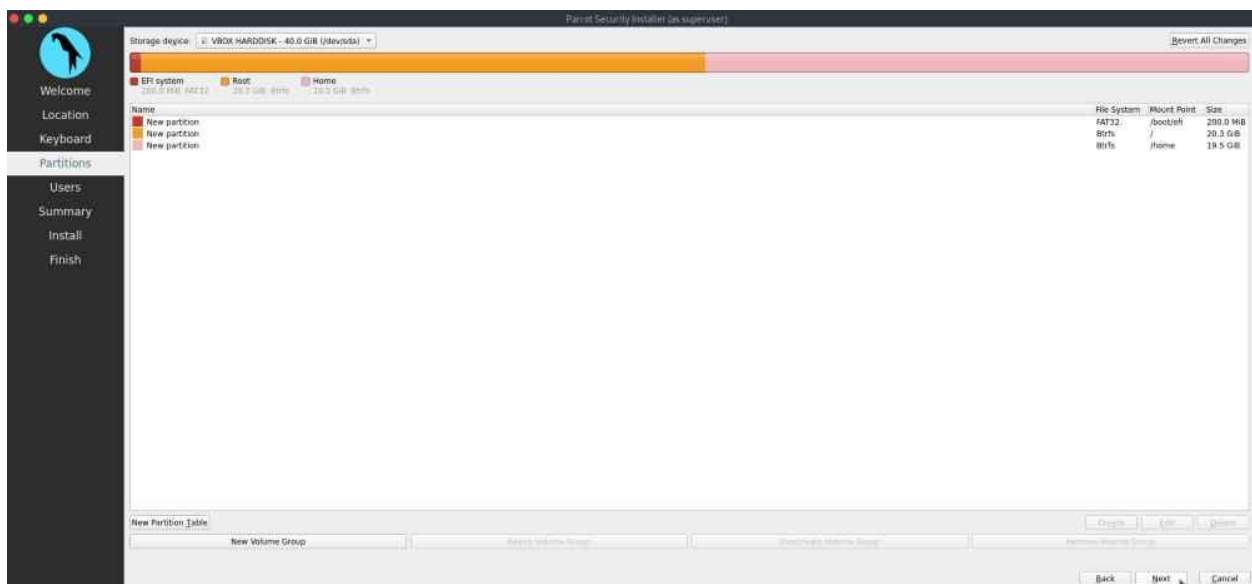
Select it and click on Create:



As the setup window appears, setup File System as Btrfs and Mount Point as /home. When finished, click on Ok:



Now the partitioning is completed, proceed with the installation by clicking on Next:



REFERENCES

- <https://mrw0r57.github.io/2020-05-26-parrot-os-basics-1-3/>
- <https://www.scaler.com/topics/operating-system/process-scheduling/>
- https://www.javatpoint.com/cpu-scheduling-algorithms-in-operating-systems?fbclid=IwAR2-f0PhTega82ETyPO_aA6P83poLhQFw3gcQvAiW3DzCvNxND67Xfkc1rvk
- <https://www.turing.com/kb/different-types-of-non-preemptive-cpu-scheduling-algorithms>
- <https://www.geeksforgeeks.org/memory-management-in-operating-system/>
- <https://www.javatpoint.com/linux-memory-management>
- <https://www.indeed.com/career-advice/career-development/what-is-storage-management#:~:text=Storage%20management%20entails%20the%20processes, costs%2C%20performance%20and%20storage%20capacity.>
- https://parrotsec.org/docs/configuration/file-and-directory-permissions/?fbclid=IwAR0Y5w_7uRRdeRG0-4S82xGalrR1joOd0kzy6LSQKdcwxkSI9VyRmm-NjVI
- [https://www.techtarget.com/whatis/definition/input-output-I-O#:~:text=I%2FO%20\(input%2Foutput\)%2C%20pronounced%20%22eye, hard%20disks%2C%20keyboards%20and%20mice.](https://www.techtarget.com/whatis/definition/input-output-I-O#:~:text=I%2FO%20(input%2Foutput)%2C%20pronounced%20%22eye, hard%20disks%2C%20keyboards%20and%20mice.)
- <https://pmihaylov.com/standard-io/>
- <https://parrotsec.org/docs/installation/manual-installation/>
- <https://www.techtarget.com/searchstorage/definition/file-system#:~:text=In%20a%20computer%2C%20a%20file, difficult%20to%20identify%20and%20retrieve.>

