# Linux

## Introduction

**It's an open-source operating system  
it means that everyone can access the code and make changes  
its lighter and more reliable than windows (and free)**

**·       אנחנו עובדים עם מערכת ניהול קבצים בשם ext.**

**·       אנחנו משתמשים ב shell שנקרא bash. הוא גרסה מחודשת של sh.**

### Terminal

**דברים שכדאי לדעת**

#### **·       נתיב יחסי: יחסית לאן שאני עכשיו –לא יתחיל ב " /" - אתחיל ממה שאני רואה בתוך התיקיה**

**·       נתיב אבסולוטי: לא משנה איפה אני נמצאת עכשיו (לא צריך cd) תמיד יתחיל ב " /"**  **מקביל ל "C:" ב windows**

**·       ניתן לקצר תהליכים באמצעות " \* " לצורך תפיסת כל הקבצים תחת תיקיה מסוימת**

**·       אין דבר כזה גיבוי בלי שחזור שעובד!**

**·       כל פקודה שאריץ שמורה בתור קובץ ב /usr/bin**

**·       יוזר רוט לא צריך סיסמאות כדי להיכנס ליוזרים רגילים**

**·       קבצים שמתחילים ב "." הם קבצים נסתרים (לא נראה אותם ב ls)**

**·       Bashrc= קובץ שמכיל הגדרות מערכת, זה שיושב ב / משפיע על כולם וזה שיושב אצל היוזר משפיע רק על היוזר**

**·       \w = [a-zA-a0-9] (גם רווחים)**

#### Commands

|  |  |
| --- | --- |
| **Command** | **Action** |
| **man** | **מדריך על השימוש בפקודה** |
| **whoami** | **מציג את שם המשתמש** |
| **Id** | **פרטים על המשתמש** |
| **pwd** | **מראה את הנתיב המלא שאני נמצאת בו** |
| **Ls** | **מציג את כל התיקיות והקבצים בנתיב** |
| **-** | **flag** |
| **touch** | **יוצר קובץ גנרי ריק** |
| **mkdir** | **יצירת תיקיה** |
| **rmdir** | **מחיקת תיקיה** |
| **rm** | **מחיקת קובץ** |
| **tree** | **מציג את עץ התיקיות בנתיב** |
| **cp** | **העתק הדבק** |
| **mv** | **גזור הדבק** |
| **cd "path"** | **מעבר בין נתיבים** |
| **..** | **תיקיה אחת למעלה** |
| **.** | **התיקיה שאני נמצאת בה** |
| **cd / cd ~** | **חזרה לתיקיית הבית של היוזר** |
| **clear / ctrl + l** | **מנקה את הטרמינל** |
| **.\"file\_name"** | **הרצה של קובץ** |
| **nano** | **עורך בסיסי** |
| **חץ למעלה** | **מחזיר לי את כל הפקודות שעשיתי** |
| **cat** | **מדפיס את התוכן של הקובץ** |
| **more** | **מראה באחוזים כמה נשאר** |
| **less** | **פותח במסך נפרד גדול יותר** |
| **head** | **מדפיס את השורות הראשונות** |
| **tail** | **מדפיס את השורות האחרונות** |
| **su - "user\_name"** | **מעבר למשתמש אחר** |
| **su -** | **מעבר למשתמש root** |
| **sudo** | **ביצוע פעולה באמצעות משתמש root** |
| **passwd** | **מתן סיסמה ליוזר (צריך להיות בתוך היוזר)** |
| **usermod** | **הוספת יוזר לקבוצה** |
| **exit / ctrl + d** | **חזרה לאן שהייתי** |
| **apt install " Package\_name"** | **פקודה להתקנה** |
| **tar** | **פקודת דחיסה/כיווץ** |
| **grep "word" "path"** | **מראה רק את השורות שמכילות מילה מסוימת** |
| **grep ^"letter" "path"** | **כל המילים שמתחילות באות מסוימת** |
| **grep "letter"$ "path"** | **כל המילים שמסתיימות באות מסוימת** |
| **which** | **מראה את הנתיב שבו נמצאת הפקודה** |
| **$** | **קורא למשתנה** |
| **" "** | **הופך את התוכן לסטרינג אחד (כשיש $)** |
| **' '** | **מציג את הסטרינג בדיוק כמו שהוא כתוב(כשיש $)** |
| **path** | **משתנה שמכיל את הנתיבים לכל הפקודות** |
| **useradd** | **פקודה להוספת משתמש** |
| **wc** | **ספירת מילים** |
| **wc -l** | **ספירת שורות** |
| **Groupadd** | **יצירת קבוצה** |
| **Groups** | **מראה את כל הקבוצות שהיוזר נמצא בהן** |
| **>** | **זורק את ה output על המסך לתוך הקובץ** |
| **Nano ."file\_name"** | **גישה ועריכה של קובץ נסתר** |
| **sort** | **מסדר את הפלט מגדול לקטן** |
| **awk** |  |
| **sed** |  |
| **chown** | **משנה בעלים** |
| **trap** | **לתפוס סגמנטים** |
| **chmod** | **שינוי הרשאות על תיקייה** |

#### Link to [Additional commands](https://www.geeksforgeeks.org/basic-shell-commands-in-linux/)

#### Commands Examples

**·       mv "file\_name" "dir\_name"/"new\_file\_name" = יעביר את הקובץ לתיקיה אחרת עם שם חדש**

**·       mv "old\_file\_name" "new\_file\_name" = שינוי שם קובץ**

**·       mv "path/source\_folder" "path/destination\_folder" = העתקת תיקיה לנתיב חדש**

**·       cp "file\_name" "dir\_name" = יעתיק את הקובץ לתיקיה אחרת**

**·       cp –r "old\_dir\_name" "new\_dir\_name" = יצירת עותק של תיקיה עם שם חדש**

**·       tar –zcvf "file\_name.tar.gz" "file\_name" = zipדחיסת תיקיה לקובץ**

**·       useradd "user\_name" –m –s /bin/bash = bashיצירת יוזר ותיקיית בית עם**

**·       usermod –aG "group\_name" "user\_name"= הוספת יוזר לקבוצה**

**·       awk –F: '{ print $1}' /etc/passwd**

**·       sed 's/"old\_value"/"new\_value"/g' > "file\_name"= משנה את כל המילים בטקסט למילה אחרת ושומר לקובץ**

**Useradd user2 -m -b /usr/bin/bash**

#### Flags

**-l = long: הפלט יהיה מפורט יותר**

**-t = time:  מראה לפי הזמן**

**-r = reversed: הפלט יהיה בסדר הפוך**

**-R = recursive: יראה גם את כל התיקיות והקבצים שמתחת**

**-"number": מדפיס מספר מסוים של שורות**

**-c= compressדחיסה :**

**-z= zip:  סוג הארכיון**

**-f=file: הקובץ שאליו זה יכנס (מיד אחריה יגיע שם הקובץ)**

**-v= verbose:  ידפיס את הפעולות שהוא עושה**

**-x= extract: חילוץ**

**-s= shell בחירת**

**-l = מראה את כל ההרשאות של כל תיקייה**

**ניתן לאחד פלגים. דוגמה: -ltr**

#### Apt command

#### Stands for Advanced Package Tool

#### You can use the apt command to install, delete or remove apps

##### Install examples

**vim**

**spell**

**changeme**

**tree**

#### nano text editor

##### Opening and Creating Files:

##### nano "file\_name"

**The above command will open a new file with filename as shown in the output. In case the file already exists it will open the same and in case the file is not there in the current directory it will create a new one.**

##### Saving and Exiting:

##### To save your changes press Ctrl+o.

**To exit nano press Ctrl+x**

**It will ask you for the filename. In case you want to save the changes to a new file or want to create a new file then change the name. else, keep the name same.**

##### Copping, cutting, and pasting:

##### Ctrl+k is used to cut and Ctrl+u is used to paste the text.

##### To search a word in a file:

##### To search Press Ctrl+w

#### To move to the next match, press Alt+w

#### Variables

**I can define a variable and then call it in a script**

**By convention, Unix shell variables will have their names in UPPERCASE**

##### Defining and Accessing Variables

### To define a variable: Variable\_name="variable\_value"

### To access the value stored in a variable: “$variable\_name”

**Sudo problem - permission**

**Apt install sudo**

**usermod -aG sudo gihanroey**

**Sudo visudo        -   etc/sudoers**

Directories:

- The file system is a labyrinth of directories that can be navigated using the change directory command. /

- Use the **'cd'** command followed by the directory name to navigate through directories.

- The **'bin'** directory contains essential system binaries like 'gzip', 'curl', and 'ls'.

- The '**s-bin**' directory contains system binaries that should only be executed by the root user.

- The **'lib'** directory stores common libraries that are shared by binaries in the 'bin' and 's-bin' directories.

- The **'usr'** directory contains non-essential binaries or applications intended for end-users.

- The **'local'** directory under 'user' provides a safe place to store binaries that you compile manually.

- The **'etc'** directory contains editable text configuration files, such as '.conf' files, for customising the behaviour of software.

## Navigating Through the File System

To navigate through the file system, we use the \"ls\" and \"cd\" commands. \"ls\" lists all the contents of our current working directory, while \"cd\" changes directories. For example, \"cd Desktop\" will take us to the desktop directory. To go back to the previous directory, we use \"cd ..\". If we keep using this command, we will eventually reach the root of the file system, represented by \"/\".

## Understanding Linux Files

In Linux, everything is a file, from configuration files to devices like hard drives and printers. Even the commands we use are files, stored in the \"bin\" directory. We can access this directory by typing \"cd bin\". Inside this directory, we will find all the command binaries, such as \"ls\" and \"cd\".

## An Exploration of Linux Directories and File Structure

The root directory is the parent directory that holds all the other directories in the Linux file system. It contains essential directories like bin, dev, etc., home, and var.

-  /bin and /usr/bin: /bin directory contain essential binary files used by the system and users to execute commands, while the /usr/bin directory contains user binaries.

-  /dev: The /dev directory contains device files for different hardware, including printers, hard drives, and CD-ROMs.

-  /etc: The /etc directory contains system configuration files, which set up system-wide and user-specific configurations.

-  /home: The /home directory contains home directories for all users on the system.

-  /lib and /usr/lib: The /lib directory contains shared library files used by the system, while the /usr/lib directory contains shared libraries used by applications installed on the system.

-  /proc: The /proc directory contains information about the system's processes.

-  /root: The /root directory contains the home directory for the root user.

Users and Groups-

#### bashrc-

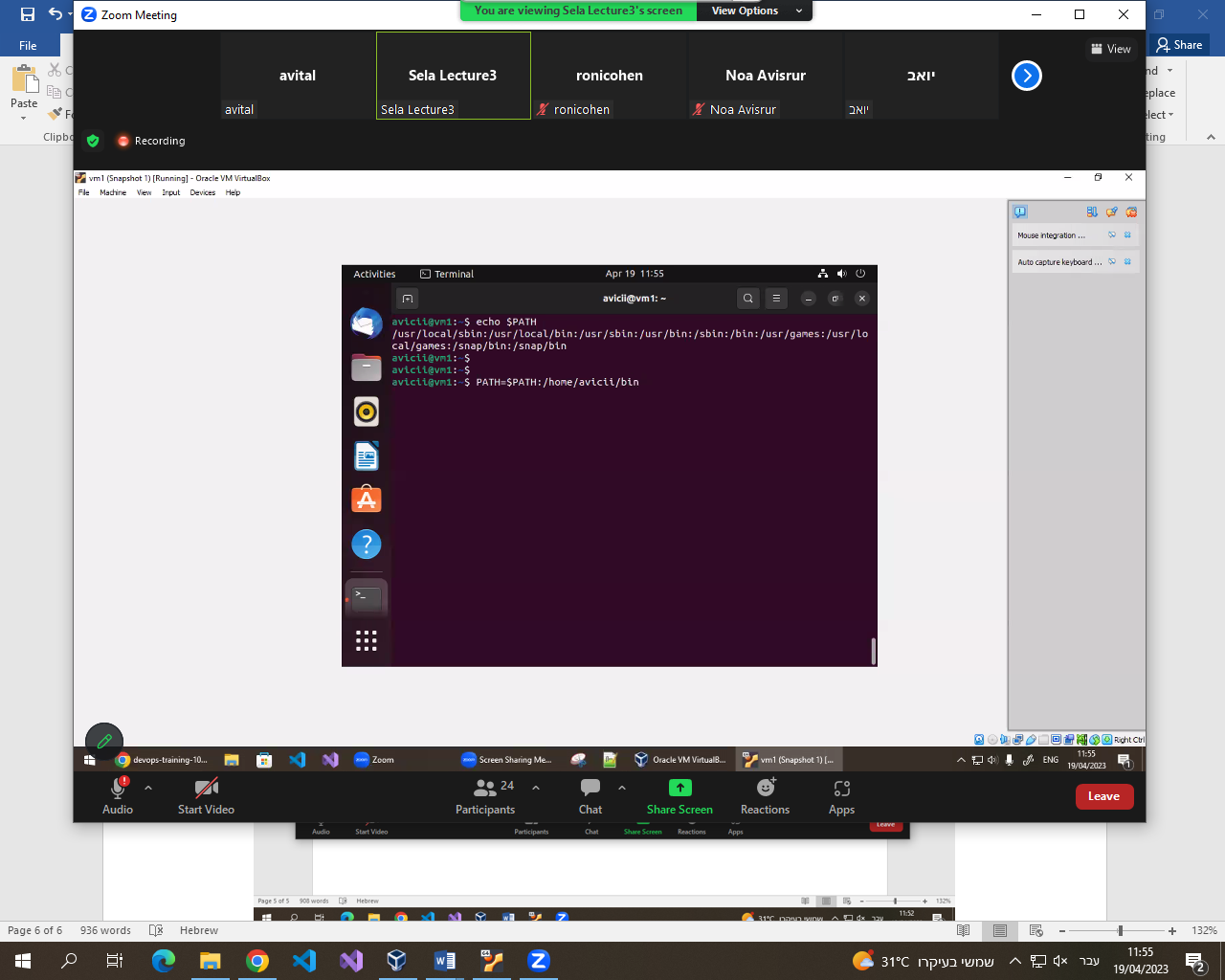
Bashrc= קובץ שמכיל הגדרות מערכת.  
 יש אחד על תיקיית הבית של היוזר שמשפיע רק על היוזר.  
 מגיעים אליו באמצעות: nano .bashrc  
 ויש אחד שיושב ב/etc/bash/bash.rc  והוא משפיע על כל היוזרים.

### Alias

רשימה של ביטויים שעוזרים לקצר פעולות ארוכות, ניתן לראות אותם תחת alias.  
 דוגמה:  
 מוסיפים alias חדש בקובץ bashrc.

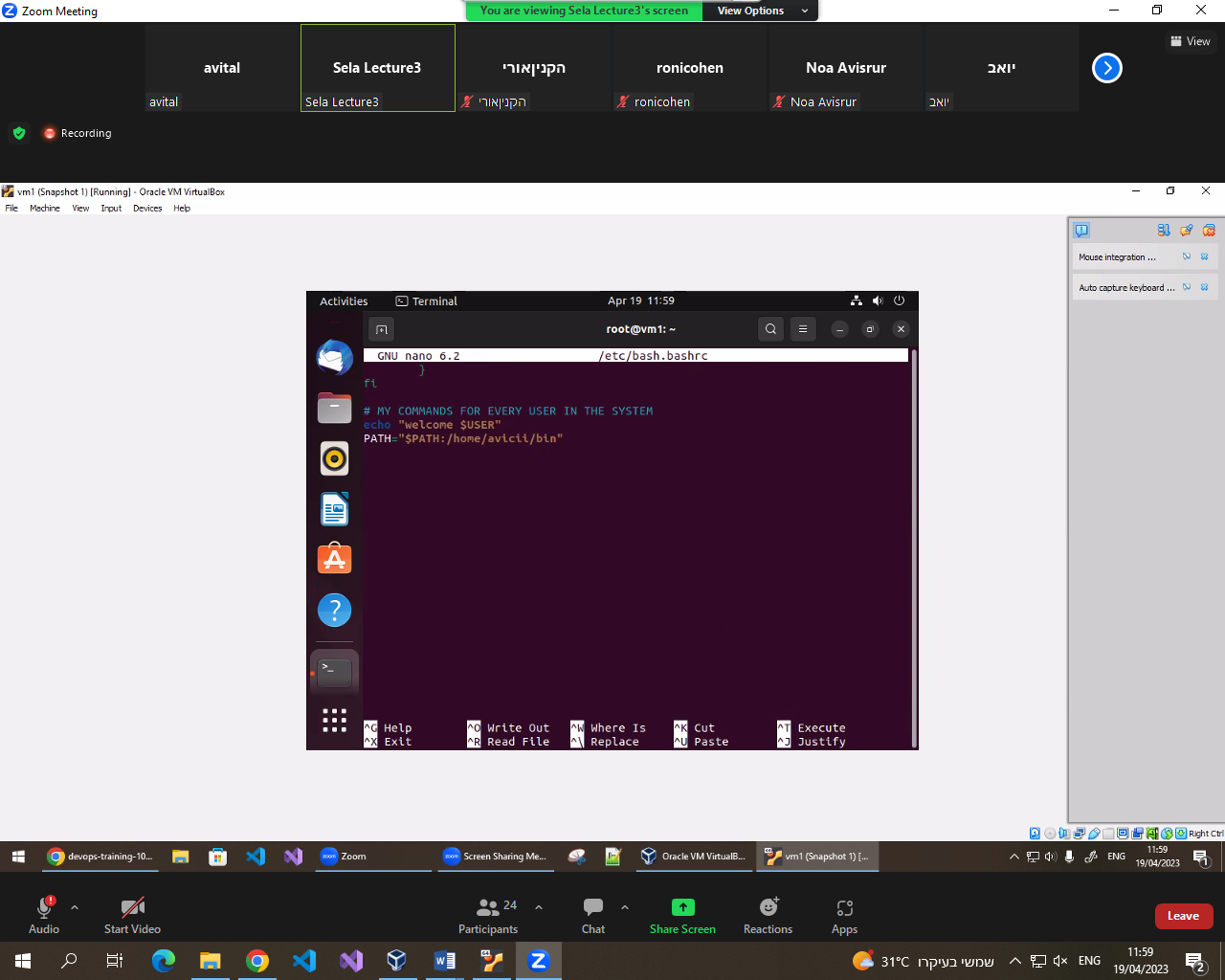
### Path

משתנה שמחזיק את כל הנתיבים שבהם יושבות פקודות שאפשר להריץ.  
 אם הנתיב של הפקודה לא יושב במשתנה הזה אני לא אוכל להריץ אותה.  
 הוא יושב ב /etc/environment



ניתן להוסיף נתיב נוסף ב /etc/environment  
 או באמצעות (למרות שיש שם איזה באג שזה יוסיף את אותה השורה פעם נוספת בכל התחברות)

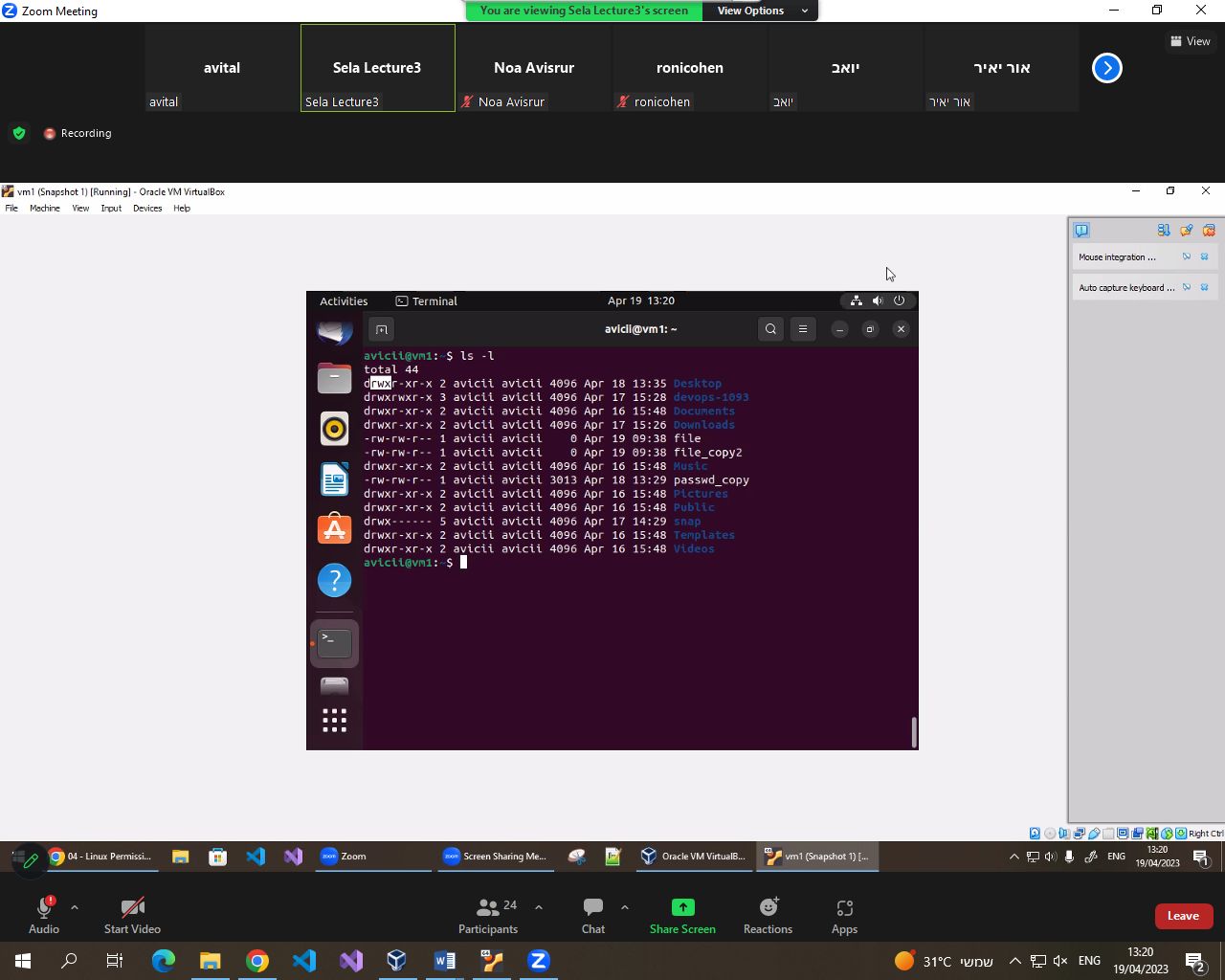
Nano /etc/bash/bashrc



### Profile

#### Permissions

כדי לראות את ההרשאות של האובייקטים נעשה: ls -l



D =directory

- = file

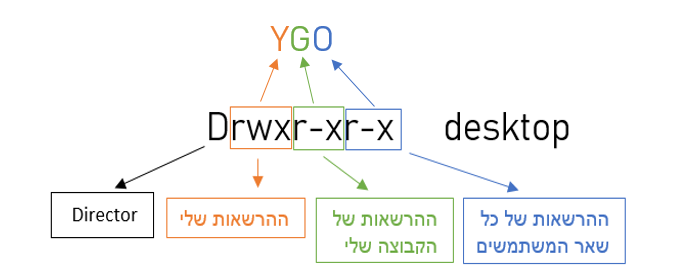
R = read

W = write

X = execute

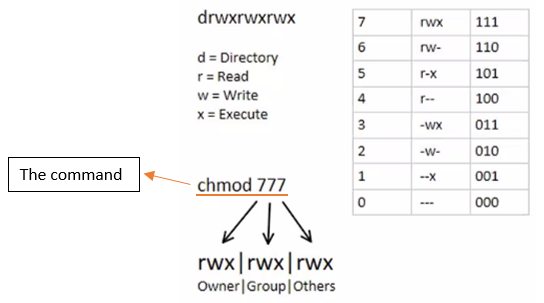
עבור כל אובייקט יש 3 סוגי הרשאות: משמאל לימין: אני, הקבוצה שלי, כל השאר

דוגמא-



* לכל תיקיה יש יוזר שהוא האונר שלה (בדרך כלל מי שיצר אותה) וגרופ שהוא האונר שלה.

מתן הרשאות:



## chmod Modifies File Permissions

**Chmod** takes three main arguments: r, w, and x, which stand for read, write, and execute, respectively. Adding or removing combinations of the arguments controls file and folder permissions. For example, chmod +rwx adds permission to read, write, and execute scripts. Running chmod -wx removes the ability to write and execute

The “who” values we can use are:

* *u*: User, meaning the owner of the file.
* *g*: Group, meaning members of the group the file belongs to.
* *o*: Others, meaning people not governed by the u and g permissions.
* *a*: All, meaning all of the above.

If none of these are used, chmod behaves as if “a” had been used.

The “what” values we can use are:

* *–*: Removes the permission.
* *+*: Grants the permission. The permission is added to the existing permissions. If you want to have this permission and only this permission set, use the = option, described below.
* *=*: Set a permission and remove others.

The “which ” values we can use are:

* *r*:  The read permission.
* *w*: The write permission.
* *x*: The execute permission.

Ex- We want the user dave to have read and write permissions and the group and other users to have read permissions only. We can do using the following command:

chmod u=rw,og=r new\_file.txt

Using the “=” operator means we wipe out any existing permissions and then set the ones specified.

chmod +x filename

sudo chgrp “root” filename

recursively change the group ownership of a folder and all of its contents. -R

sudo chgrp -R geeksforgeeks GFG

Useradd-

When executed without any option, useradd creates a new user account using the default settings specified in the /etc/default/useradd file

The command adds an entry to the /etc/passwd, [/etc/shadow,](https://linuxize.com/post/etc-shadow-file/) /etc/group and /etc/gshadow files.

To be able to log in as the newly created user, you need to set the user password. To do that run the [passwd](https://linuxize.com/post/how-to-change-user-password-in-linux/) command followed by the username:

On most Linux distributions, when creating a new user account with useradd, the user’s home directory is not created.

Use the -m (--create-home) option to create the user home directory as /home/username:

$ sudo useradd -m username

By default useradd creates the user’s home directory in /home. If you want to create the user’s home directory in other location, use the d (--home) option.

Here is an example showing how to create a new user named username with a home directory of /opt/username:

sudo useradd -m -d /opt/username username

invoke useradd with the -u (--uid) option to create a user with a specific UID. For example to create a new user named username with UID of 1500 you would type:

sudo useradd -u 1500 username

You can verify the user’s UID, using the [id](https://linuxize.com/post/id-command-in-linux/) command:

id -u username

When creating a new user, the default behavior of the useradd command is to create a group with the same name as the username, and same GID as UID.

The -g (--gid) option allows you to create a user with a specific initial login group. You can specify either the group name or the GID number. The group name or GID must already exist.

The following example shows how to create a new user named username and set the login group to users type:

sudo useradd -g users usernameCopy

To verify the user’s GID, use the id command:

id -gn username

## editing sudoers permissions-

The users’ and groups’ sudo privileges are defined in the /etc/sudoers file. Adding the user to this file allows you to grant customized access to the commands and configure custom security policies.

Alternatively, we can add the user to the sudo group using “usermod”:

(usermod command or modify user is a command in Linux that is used to change the properties of a user in Linux through the command line)

sudo usermod -aG sudo <user>

This will add <user> to the sudo group. Members of the sudo group have full sudo privileges.

visudo uses vim to open the /etc/sudoers

Let’s say you want to allow the user to run sudo commands without being asked for a password. To do that, open

the /etc/sudoers file:

<user> ALL=(ALL) NOPASSWD: <commands>

This command will show all of the system’s users names-

cat /etc/passwd | cut -d: -f1

## What Is Linux PATH?

When a user invokes a command in the terminal, the system executes a program. Therefore, Linux has to be able to locate the correct executable. **PATH** specifies program directories and instructs the system where to search for a program to run.

### How to View the Directories in PATH

To print all the configured directories in the system's **PATH** variable, run the [echo command](https://phoenixnap.com/kb/echo-command-linux):

echo $PATH

Furthermore, running [which](https://phoenixnap.com/kb/which-command-linux) on a certain command shows **where its executable** is. For instance, execute **which** with **whoami**:

which whoami

**Temporarily** adding a directory to **PATH** affects the current terminal session only. Once users close the terminal, the directory is removed.

To temporarily add a directory to **PATH**, use the [**export**](https://phoenixnap.com/kb/linux-export-command)**PATH** command:

export PATH="/Directory1:$PATH"

Add a directory to **PATH** **permanently** by editing the [.bashrc file](https://phoenixnap.com/kb/bashrc) located in the **Home**  directory**.** Follow these steps:

1. Open the **.bashrc** file using a text editor. The example below uses [Vim](https://phoenixnap.com/kb/vim-commands-cheat-sheet).

Opening .bashrc in Vim.

2. Go to [the end of the file](https://phoenixnap.com/kb/vim-go-to-end-of-file).

3. Paste the export syntax at the end of the file.

export PATH="/Directory1:$PATH"

4. Execute the script or reboot the system to make the changes live.

Editing the **.bashrc** file adds a directory for the current user only. To add the directory to the **PATH** for all users, edit the **.profile** file:

If the directory export string was added to the **.bashrc** or **.profile** file, remove it using the same method. Open the file in a text editor, navigate to the end of the file, and remove the directory.

To remove a directory from **PATH**, use string replacement:

export PATH=${PATH/'/Directory1'/}

## What is an inode?

An inode is **a data structure that keeps track of all the files and directories within a Linux or UNIX-based filesystem**. So, every file and directory in a filesystem is allocated an inode, which is identified by an integer known as “inode number”. These unique identifiers store metadata about each file and directory.

**All inodes within the same filesystem are unique**. However, the same inode number can be used in different filesystems. Because the filesystem ID and each inode number are combined to create unique identification labels.

### Metadata stored in an inode

Inodes store metadata such as:

* File type
* File size
* Owner ID
* Group ID
* Read, write and execute permissions
* Last access time
* Last change time
* Last modification time

As explained above, **each inode is identified by an inode number**. Therefore, when creating or copying a file, Linux assigns a different inode number to the new file. However, when moving a file, the inode number will only change if the file is moved to a different filesystem. This applies to directories as well.

You can check the number of inodes in your filesystem using the **df** command with the -**i** option.

The **stat** command gives **information about the file and filesystem**. So, you can use it to check the inode number of a file.

$ stat /var/log/lastlog

You can also use the ls command, together with the -i option, to get a file’s inode number. This command **lists files and directories within the filesystem**.

[root@sta

**Link**:

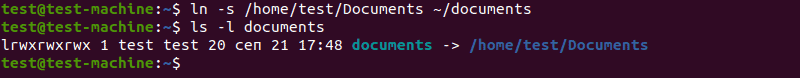
ln -s test\_file.txt link\_file.txt

This creates a symbolic link **(link\_file.txt)** that points to the **test\_file.txt**.

A symbolic link can refer to a directory. To create a symbolic link to a directory in Linux: or without the -s

ln -s /mnt/external\_drive/stock\_photos ~/stock\_photos

This example creates a symbolic link named **stock\_photos** in the **home (~/)** directory. The link refers to the **stock\_photos** directory on an **external\_drive**.



If the original file is moved, deleted, or becomes unavailable (such as a server going offline), the link will be unusable. To remove a symbolic link, use either the **rm** (remove) or **unlink** command:

rm link\_file.txt

unlink link\_file.txt

**Soft Links (Symbolic Links):**

- Soft links are created using the "ln -s" command and create a new file that acts as a symbolic pointer to the original file or directory.

- Soft links can be created across different file systems and partitions, as they do not depend on the underlying file system's inode number. This allows for greater flexibility in managing files and directories.

- Soft links are lightweight, meaning they take up very little disk space as they only contain the path to the original file.

- If the original file is deleted or moved, the soft link will still exist but will be broken and point to a non-existent file. This can cause issues if any programs or scripts rely on the soft link.

- Soft links can be easily identified by the "l" symbol in the file permissions.

**Hard Links:**

- Hard links are created using the "ln" command and create an additional link to the original file or directory that points to the same inode.

- Hard links can only be created within the same file system or partition, as they rely on the underlying file system's inode number to point to the original file.

- Hard links are essentially the same file, just with multiple names. Any changes made to the original file will be reflected in all hard links that point to it.

- Hard links take up very little disk space as they do not create a new file, but instead just create an additional directory entry pointing to the original file.

- If the original file is deleted, the hard link will still exist and the data will be retained until all hard links to the file are deleted.

- Hard links can be identified by the absence of the "l" symbol in the file permissions.

In summary, soft links act as a symbolic pointer to the original file or directory and can be created across different file systems, while hard links create an additional directory entry pointing to the same inode and can only be created within the same file system or partition. Both types of links have their uses and it's important to understand the differences to determine which type is appropriate for your needs.

* A file in any Unix-based operating system comprises of the data block(s) and an inode. The data blocks store the actual file contents. On the other hand, an inode store file attributes (except the file name) and the disk block locations.
* Ls -i (inode check)
* **A hard link is just another file that points to the same underlying inode as the original file.** And so, it references to the same physical file location.
* **Stat “folder” -** gives me information (blocks, lincks, inode)

As a user, if there is an alias that you use regularly, then instead of defining it every time you open the terminal, you can save it in the .bashrc file.

For example, we can replace the whoami command with the following line of code.

alias wmi='whoami'

Copy

Don’t forget to save the edit and then run:

$ source .bashrc

Copy

Now I can use wmi command and the terminal will run it as whoami.

Filesystem-

In Linux, a filesystem is essentially a method of organizing and managing data on a storage device, such as a hard drive or a USB stick. It provides a way to store and access files, directories, and other data in a structured and organized way.

In Linux, the most commonly used filesystem is the ext4 filesystem. This filesystem is known for its reliability, scalability, and performance. However, other filesystems can also be used in Linux, depending on the specific requirements of the system.

The Linux filesystem hierarchy is organized in a tree-like structure, with the root directory at the top of the tree. Below the root directory are various subdirectories, such as /bin, /usr, /home, and so on. Each directory can contain files, as well as other subdirectories.

Files and directories in a Linux filesystem are identified by a unique path, which starts at the root directory and specifies each subdirectory leading to the file or directory in question. For example, the path /usr/share/doc would refer to the "doc" directory located within the "share" directory, which in turn is located within the "usr" directory.

**In addition to providing a structured way to store and access files, Linux filesystems also include features such as permissions, ownership, and security. These features allow administrators to control who can access specific files and directories, and what actions they can perform on those files and directories.**

Overall, the Linux filesystem is a fundamental part of the operating system, providing a reliable and flexible way to manage data on storage devices.

## [Types of Processes in Linux](https://www.digitalocean.com/community/tutorials/process-management-in-linux#types-of-processes-in-linux)

In Linux processes can be of two types:

* **Foreground Processes**  
  depend on the user for input  
  also referred to as interactive processes
* **Background Processes**  
  run independently of the user  
  referred to as non-interactive or automatic processes

**Parent and Child process :** The 2nd and 3rd column of the ps –f command shows process id and parent’s process id number. For each user process, there’s a parent process in the system, with most of the commands having shell as their parent.

**Zombie and Orphan process :**After completing its execution a child process is terminated or killed and SIGCHLD updates the parent process about the termination and thus can continue the task assigned to it. But at times when the parent process is killed before the termination of the child process, the child processes become orphan processes, with the parent of all processes “init” process, becomes their new pid.   
A process which is killed but still shows its entry in the process status or the process table is called a zombie process, they are dead and are not used.

**Daemon process :**They are system-related background processes that often run with the permissions of root and services requests from other processes, they most of the time run in the background and wait for processes it can work along with for ex print daemon.   
When **ps –ef** is executed, the process with ? in the tty field are daemon processes.

### [Process States in Linux](https://www.digitalocean.com/community/tutorials/process-management-in-linux#process-states-in-linux)

A process in Linux can go through different states after it’s created and before it’s terminated. These states are:

* **Running**
* **Sleeping**
  + **Interruptible sleep**
  + **Uninterruptible sleep**
* **Stopped**
* **Zombie**
* A process in **running** state means that it is running or it’s ready to run.
* The process is in a **sleeping** state when it is waiting for a resource to be available.
* A process in **Interruptible sleep** will wakeup to handle signals, whereas a process in **Uninterruptible sleep** will not.
* A process enters a **stopped** state when it receives a stop signal.
* **Zombie** state is when a process is dead but the entry for the process is still present in the table.
* **Deamon – a process that always runs.**

## [Different Commands for Process Management in Linux](https://www.digitalocean.com/community/tutorials/process-management-in-linux#different-commands-for-process-management-in-linux)

There are two commands available in Linux to track running processes. These two commands are **Top** and **Ps**.

### [1. The top Command for Mananging Linux Processes](https://www.digitalocean.com/community/tutorials/process-management-in-linux#1-the-top-command-for-mananging-linux-processes)

To track the running processes on your machine you can use the **top** command. (like task manager) or **htop** (better)

$ top

Let’s understand the output a little better:

* **PID**: Unique Process ID given to each process.
* **User**: Username of the process owner.
* **PR**: Priority given to a process while scheduling.
* **NI:** ‘nice’ value of a process.
* **VIRT**: Amount of virtual memory used by a process.
* **RES**: Amount of physical memory used by a process.
* **SHR**: Amount of memory shared with other processes.
* **S**: state of the process
  + ‘D’ = uninterruptible sleep
  + ‘R’ = running
  + ‘S’ = sleeping
  + ‘T’ = traced or stopped
  + ‘Z’ = zombie
* **%CPU**: Percentage of CPU used by the process.
* **%MEM**; Percentage of RAM used by the process.
* **TIME+:** Total CPU time consumed by the process.
* **Command**: Command used to activate the process.

**Ps** command is short for ‘Process Status’. It displays the currently-running processes.

$ ps

While ps command only displays the processes that are currently running, you can also use it to list all the processes.

$ ps -A

(This command lists even those processes that are currently not running.)

To stop a process in Linux, use the '**kill’** command.

There are different types of signals that you can send. However, the most common one is ‘kill -9’ which is ‘**SIGKILL**’.

You can list all the signals using:

$ kill -L

The default signal is 15, which is **SIGTERM**. Which means if you just use the kill command without any number, it sends the SIGTERM signal.

The syntax for killing a process is:

$ kill [pid]

## **$ jobs 1,2…… little slots I can use when free with a % sign**

Ppid – parent process id

Init – the very first process, the father of all the processes

## **Other process commands:**

**bg**: A job control command that resumes suspended jobs while keeping them running in the background. ^z  
Syntax:

The character % introduces a job specification. The Job can be a process ID (PID) number

**bg [ job ]**

For example:

bg %19

or to finish a process with **“&”** it initially runs it in the background.

**fg**: It continues a stopped job by running it in the foreground.   
Syntax:

**fg [ %job\_id ]**

For example

fg 19

**nice**: It starts a new process (job) and assigns it a priority (nice) value at the same time.   
Syntax:

nice [-nice value]

nice value ranges from -20 to 19, where -20 is of the highest priority.

**renice** : To change the priority of an already running process renice is used.   
Syntax:

renice [-nice value] [process id]

**df**: It shows the amount of available disk space being used by file systems   
Syntax:

df

**free**: It shows the total amount of free and used physical and swap memory in the system, as well as the buffers used by the kernel   
Syntax:

free

$ ps -ef | grep sleep

Kill “PID”

#### Jobs

**trap** allows you to catch signals and execute code when they occur. Signals are asynchronous notifications that are sent to your script when certain events occur. Most of these notifications are for events that you hope never happen, such as an invalid memory access or a bad system call. However, there are one or two events that you might reasonably want to deal with. There are also "user" events available that are never generated by the system that you can generate to signal your script. Bash also provides a psuedo-signal called "**EXIT**", which is executed when your script exits; this can be used to make sure that your script executes some cleanup on exit.

The signals most commonly used with the **trap** command are:

* **SIGHUP (1)** - Clean tidy-up
* **SIGINT (2)** - Interrupt
* **SIGQUIT (3)** - Quit
* **SIGABRT (6)** - Cancel
* **SIGALRM (14)** - Alarm clock
* **SIGTERM (15)** - Terminate

[a command that must not be interrupted]

trap "" SIGINT SIGABRT

A typical scenario for using the **trap** command is catching the **SIGINT** signal. This signal is sent by the system when the user interrupts the execution of the script by pressing **Ctrl+C**.

The following example script prints the word "Test" every second until the user interrupts it with **Ctrl+C**. The script then prints a message and quits.

trap echo “The script is terminated; exit" SIGINT

while true

do

echo Test

sleep 1

done

**sudo apt update/upgrade !!!**

**Fuser**- To list out all processes associated with a file or directory, just mention the filename or path of the directory or file with the fuser command. Here is one example:

fuser.

Or

fuser ~/

### For verbose output, use -v

The output of the previous command will only show the PIDs of the process, which very little information. To get more information about the processes, use the -v or –verbose option with the fuser command:

fuser -v .

To get the list of all processes which are using the file which is mentioned with the fuser command, use option -m or –mount. Here is one example: -m (mount)

fuser -v -m ~/.bashrc

### **Kill Processes that are Using a particular Program**

We can use the fuser command to kill the processes. To kill the process using the fuser command, use the option -k or –kill with the command and mention the path of the program.

sudo fuser -k

To kill a process interactively, use the option -i or –interactive with the above command. This command will ask while terminating each process.

sudo fuser -ki .

**lsof** command stands for **List Of Open File**. This command provides a list of files that are opened. Basically, it gives the information to find out the files which are opened by which process. With one go it lists out all open files in output console. It cannot only list common regular files but it can list a directory, a block special file, a shared library, a character special file, a regular pipe, a named pipe, an internet socket, a UNIX domain socket, and many others. it can be combined with grep command can be used to do advanced searching and listing.

**List all open files:** This command lists out all the files that are opened by any process in the system.

~$ lsof

Here, you observe there are details of files which are opened. Process Id, the user associated with the process, FD(file descriptor), size of the file all together gives detailed information about the file opened by the command, process ID, user, its size etc.

* + **FD** represents as File descriptor.
  + **cwd** : Current working directory.
  + **txt**: Text file.
  + **mem** : Memory file.
  + **mmap** : Memory mapped device.

**List all files opened by a user:** There are several users of a system and each user have different requirements and accordingly they use files and devices. To find a list of files that are opened by a specific user this command is useful.

**Syntax:**

lsof -u username

**Proc** file system (procfs) is virtual file system.

It contains useful information about the processes that are currently running, it is regarded as control and information center for kernel.

The proc file system also provides communication medium between kernel space and user space.

**Lsblk-** is used to display details about block devices and these block devices(Except ram disk) are basically those files that represent devices connected to the pc. It queries **/sys** virtual file system and **udev db** to obtain information that it displays.

Install –

$sudo apt-get install util-linux

To display block devices-

$lsblk

The **df** command (short for disk free), is used to display information related to file systems about total space and available space.

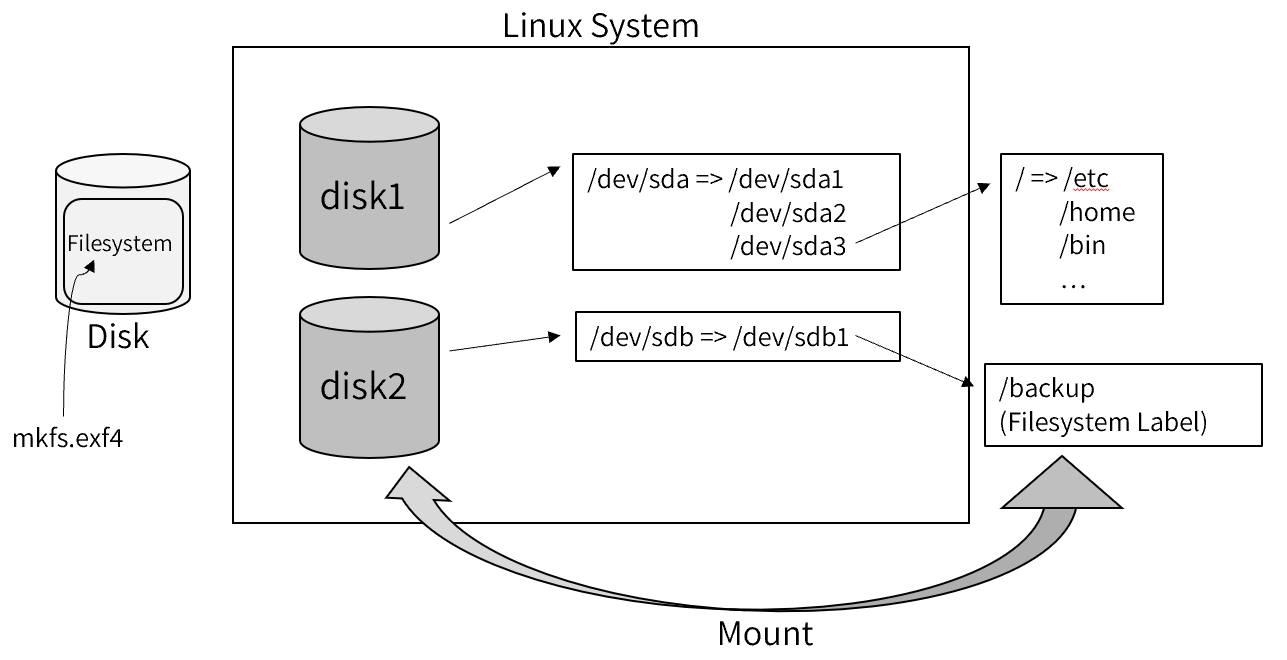
Now, if you specify particular file, then it will show mount information of that particular file.  
For example:

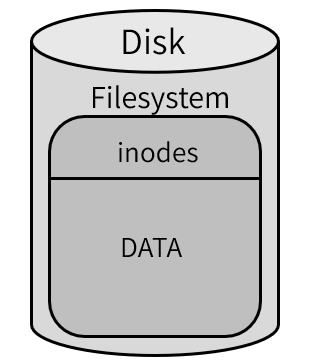
df /home/mandeep/test/test.cpp

***-a, –all :****includes pseudo, duplicate and inaccessible file systems. (all file systems)****-B, –block-size=SIZE :****scales sizes by SIZE before printing them.****-h, –human-readable :****print sizes in power of 1024****-t, –type=TYPE :****limit listing to file systems of type TYPE****-T, –print-type :****print file system type*

* lsof = מחזיר מיקום של כל התהליכים שפועל על היוזר ב-filesystem
* fuser = מחזיר את כל ה-ID של תהליכים לפי מיקום
* lsblk = מציג מידע על כל רכיבי האחסון מידע המחוברים למערכת
* df -h = אפשר לראות מידע על כל דיסק שיש לו filesystem, ולראות מי עושה לו mount
* fdisk = אפשרות ליצור ולערוך חלקי דיסק מדסיק במערכת
* mount = מקשר בין הדיסק עצמו אל ה-filesystem

mount-a עושה מאונט לכל השורות ב /etc/fstab ובודק אם הם כבר עברו מאונט או לא, אם לא אז מחזיר שגיאה





* Filesystem- מחלק דיסק ל-inodes ול-data כדי שהלינוקס ידע לעבוד עם זה
* Label- שם שמצביע על הדיסק (לא חייב להיות נתיב)
* Mount- לקחת את הדיסק ולתת לו נתיב כדי לחבר אותו לעץ קבצים, נותנים לדיסק נתיב שהמידע עליו ישמר הדיסק
* Mount אינו נשמר כשהמחשב יכבה לכן נוסיף אותו ב-/etc/fstab כדי להכניס אותו לfilesystem המוגדרים של המערכת

**fdisk** also known as format disk is a dialog-driven command in Linux used for creating and manipulating disk partition table. It is used for the view, create, delete, change, resize, copy and move partitions on a hard drive using the dialog-driven interface.   
fdisk allows you to create a maximum of four primary partitions and the number of logical partition depends on the size of the hard disk you are using. It allows the user:

* To Create space for new partitions.
* Organizing space for new drives.
* Re-organizing old drives.
* Copying or Moving data to new disks(partitions).

list the partitions on your system and see their */dev* names-

**$ sudo fdisk -l**

**View Partition on a Specific Disk:**Below command is used to view all disk partitions on device **/dev/sda**.

**$ sudo fdisk -l /dev/sda**

**View all fdisk Commands:**To see all the command which are available under fdisk command you can use **/dev/sda** partition with fdisk command.

**$ sudo fdisk /dev/sda**

(Type **m** for seeing all the operations which can perform on **/dev/sda)**

**Create a Hard Disk Partition:**For this go inside the hard drive partition that is the **/dev/sda** partition, and use the following command:

**$ sudo fdisk /dev/sda**

Now you have to type **n** to create new partition and then type **p** for making a primary partition and **e** for making an extended or logical partition depending on the type of partition to make.

Run **w** command to write the changes and reboot your system.

**5. Delete a Hard Disk Partition:**To delete a partition for the hard disk and free up space occupied by that partition for example **/dev/sdb**. Go to the command menu using following:

**$ sudo fdisk /dev/sda**

and then type **d** to go to the delete partition menu. It will prompt the partition number you want to delete

**How to view the size of your Partition:**

**$ sudo fdisk -s /dev/sda**

**Mount**-

All files in a **Linux** filesystem are arranged in form of a big tree rooted at ‘**/**‘.These files can be spread out on various devices based on your partition table, initially your parent directory is mounted(i.e attached) to this tree at ‘**/**‘, others can be mounted manually using GUI interface(if available) or using **mount** command.  
**mount** command is used to mount the filesystem found on a device to big tree structure(**Linux** filesystem) rooted at ‘**/**‘. Conversely, another command **umount** can be used to detach these devices from the Tree.

These commands tells the **Kernel** to attach the filesystem found at **device** to the **dir**.

You can use **–source** or **–target** to avoid ambivalent interpretation.

mount --target /mountpoint

* **l** : Lists all the file systems mounted yet.
* **h** : Displays options for command.
* **V** : Displays the version information.
* **a** : Mounts all devices described at **/etc/fstab**.
* **t** : Type of filesystem device uses.
* **T** : Describes an alternative fstab file.
* **r** : Read-only mode mounted.

**Mkfs.ext4 – to make a file system - mkfs.ext4 [-L name] /dev/sda**

In Linux, the "**mount**" command is used to attach a file system, such as a hard drive or USB drive, to a directory in the file system hierarchy. This makes the contents of the file system accessible to the user and programs on the system.

When you connect an external storage device to a Linux system, it needs to be mounted before it can be used. Similarly, when you create a new partition on a hard drive, you need to mount it to a directory in the file system hierarchy before you can start using it.

To mount a file system, you need to specify the device name (such as /dev/sdb1), the mount point (the directory where you want to attach the file system), and the file system type (such as ext4 or NTFS). The mount command is usually used with root privileges, either by logging in as the root user or using the sudo command.

For example, to mount a USB drive with the device name /dev/sdb1 to the directory /mnt/usb, you can use the following command:

sudo mount /dev/sdb1 /mnt/usb

Once the device is mounted, you can access its contents by navigating to the mount point directory (in this case, /mnt/usb). To unmount the device, you can use the "umount" command:

sudo umount /mnt/usb

```

It's important to properly unmount a device before disconnecting it to avoid data loss or corruption.

But you need to embed it to the system permanently, mount to /etc/fstab.

The `**du**` command in Linux is used to estimate the file space usage in a particular directory or file system. It stands for "disk usage".

The `du` command reports the amount of disk space used by a file or directory and its subdirectories. It can be useful for finding out which files or directories are using the most disk space on a system.

- To get a summary of disk usage for the current directory:

```

du -sh

```

The `-s` option displays a summary for the specified directory, and the `-h` option prints the output in human-readable format.

- To get disk usage information for a specific directory:

```

du -sh /path/to/directory

```

Replace `/path/to/directory` with the path to the directory you want to check.

- To get detailed information for all directories and files in a directory:

```

du -a /path/to/directory

```

The `-a` option displays information for all files and directories, not just directories.

- To display the disk usage of a file:

```

du -h /path/to/file

```

Replace `/path/to/file` with the path to the file you want to check.

- To display the disk usage of all directories in a file system:

```

du -sh /\*

```

This will display the disk usage of all directories in the root file system.

Note that the `du` command reports disk usage based on the file system blocks used by files and directories, not the actual size of the files. The output may therefore be slightly larger than the actual size of the files due to block allocation.

**$ ncdu**

In Linux, "**swap**" refers to the use of a dedicated space on a hard drive or SSD as virtual memory. **When the system runs out of physical memory (RAM)** for running applications, **it can use the swap space as a temporary storage** area for inactive memory pages. This allows the system to continue running without crashing due to lack of memory.

Swap space is used as a fallback mechanism when the system's physical memory is exhausted. When the system runs out of physical memory, it will start swapping out inactive memory pages to the swap space on the hard drive. This frees up physical memory for use by active processes.

Creating a swap partition or file is recommended for all Linux systems. It can significantly improve system stability and prevent crashes caused by running out of memory.

To create a swap partition, you can use a partitioning tool such as fdisk or gparted to create a new partition on the hard drive or SSD. Then, you can use the `mkswap` command to create a swap file system on the new partition.

To create a swap file, you can use the `fallocate` command to create a new file of the desired size, and then use the `mkswap` command to format it as a swap file system.

After creating a swap partition or file, you need to enable it using the `swapon` command. This will activate the swap space and make it available for use by the system.

To view the current swap usage and configuration, you can use the `swapon` or `free` command. These commands will display information about the size, usage, and status of the swap space on the system.

Swap is not essential !

A partition is a segment of memory and contains some specific data. In our machine, there can be various partitions of the memory. Generally, every partition contains a file system

#!/bin/bash

# creates users from file

# Define the path to the file containing usernames

filename="$1"

# check if user is root

if [[ $(whoami) != "root" ]]; then

echo "you must be root"

exit 1

fi

if [[ "$filename" == "" ]]

then

echo "please specify the file"

exit 1

# Read the file and create each user

while read -r username

do

# Create the user with a hd

useradd -m -s /bin/bash "$username"

# Defining the default password

password="$username@123"

# Set the password to "password" for the new user

echo "$username:$password" | chpasswd

# Print a message to confirm the user was created

echo "User $username has been created."

done < "$filename"

#!/bin/bash

#change ths ext of folders in dir

# Check if the number of arguments is correct

if [ $# -ne 2 ]; then

echo "please type-For Example: <extension> <directory>"

exit 1

fi

# Assign the arguments to variables

ext=$1

dir=$2

# Change the extension of each file in the directory

for file in "$dir"/\*;

do

# checks whether a file exists and is a regular file

if [ -f "$file" ]; then

# extract the filename

filename=$(basename -- "$file")

# extract the extension from the filename. ##\*. keeps only the ext of the filename

extension="${filename##\*.}"

# checks if the file does not have the same extension

if [ "$extension" != "$ext" ]; then

# changes the name of the file to the direct path. %.\* removes the ext of the filename

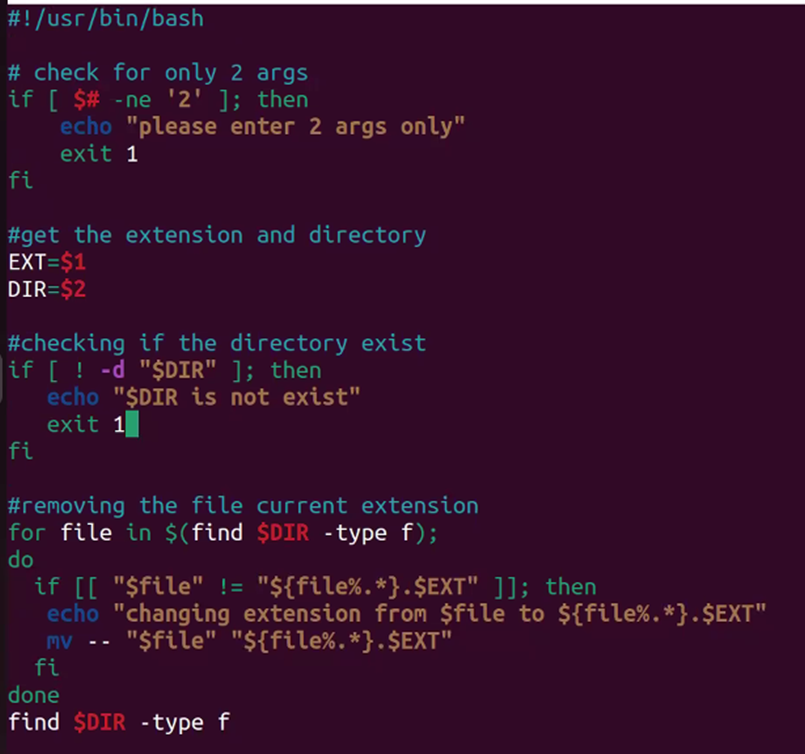
mv "$file" "$dir/${filename%.\*}.$ext"

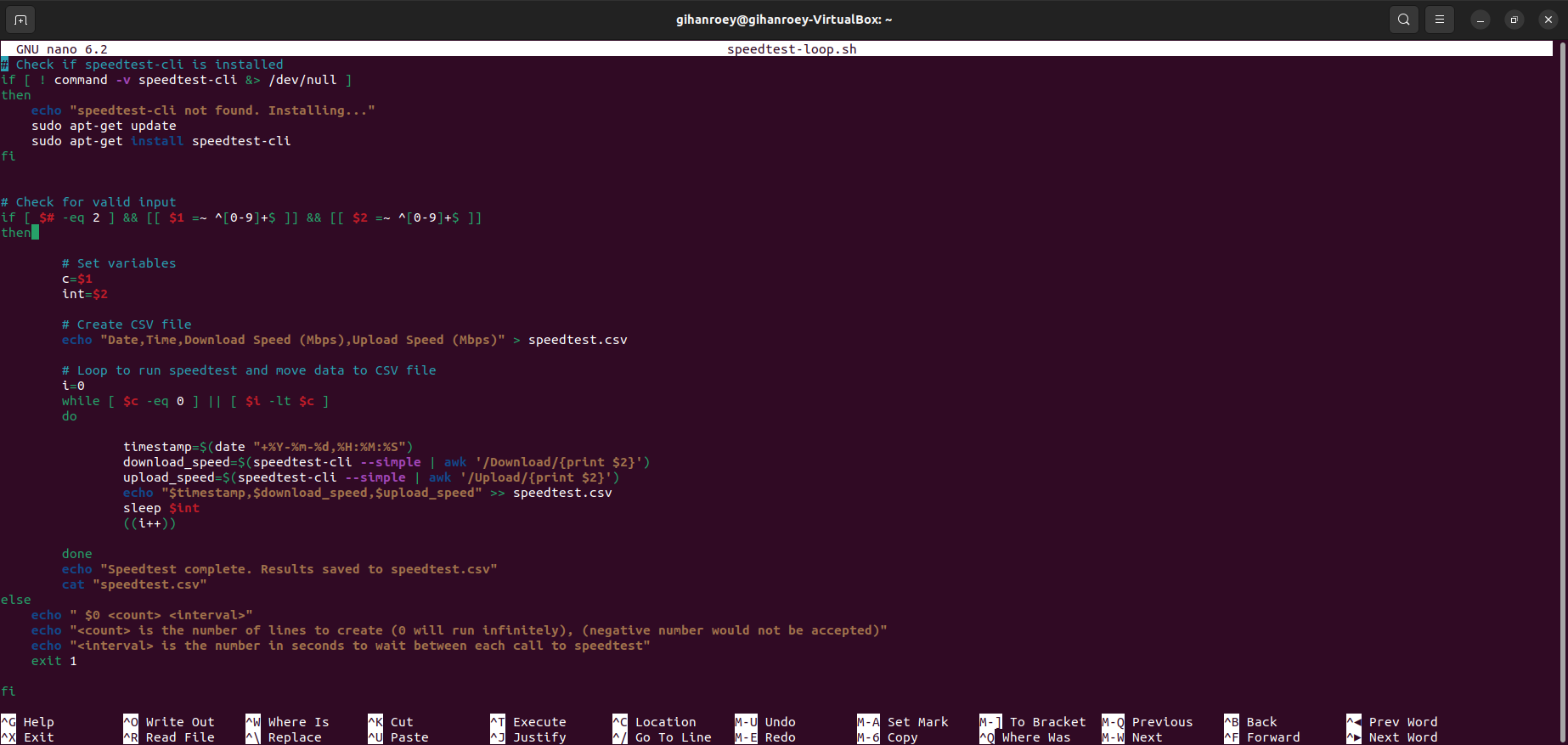
echo "the file $dir/${filename%.\*}.$ext has been changed successfuly"

fi

fi

done



speed test -

$@ - is an array. That can get infinite input.

Argument – is every phrase that comes after the names script

&>/dev/null – puts the info in there

>> - redirects an output into a file (“path”)

1>

Read – asks for input

Scripting –

https://www.hostinger.com/tutorials/bash-scripting-tutorial

-f – checks if file exists

-d – checks if directory exists ([ -d $directory ])

Bash scripting cheat sheet –

https://devhints.io/bash

**function**followed by the function name. This eliminates the need for parentheses:

function function\_name {  
first command  
second command  
}

The **for**loop runs the command for a list of items:

#!/bin/bash  
for item in [list]  
do  
[commands]  
done

The next type of loop is **while**. The script will evaluate a condition. If the condition is **true**, it will keep executing the commands until the output no longer meets the defined condition.

#!/bin/bash  
while [condition]  
do  
[commands]  
done

The last type of loop, **until**, is the opposite of **while**. It will iterate the command until the condition becomes true.

#!/bin/bash  
i=0  
until [ $i -gt 5 ]  
do   
echo $i  
((i++))  
done

In bash, `exit` is a command that is used to exit the current shell or script with a status code. The status code indicates the success or failure of the script. By convention, an exit status of `0` indicates success, and any non-zero value indicates failure.

Here are some common exit status codes and what they mean:

- $?

- `0`: Success

- `1`: General error (e.g. invalid command line arguments)

- `2`: Misuse of shell built-ins (e.g. incorrect `cd` usage)

- ‘100’ : invalid package

- `126`: Command not executable (e.g. missing execute permission)

- `127`: Command not found (e.g. incorrect command name)

- `128`: Invalid argument to exit command (e.g. non-numeric value)

- `130`: Script terminated by `Ctrl-C`

When the `exit` command is used with a status code, it immediately terminates the current shell or script and returns the specified status code to the parent process (e.g. the shell that launched the script).

In the example code `exit 1`, `exit` is used to terminate the script and return an exit status code of `1`. This is typically used to indicate that the script encountered an error, such as incorrect usage or invalid command line arguments. `exit 2` is similar, but indicates a different type of error (misuse of shell built-ins).

Csv files in bash-

**>** : write output to file (it will auto generate file if the file is not exist) and run over existing data

>> : appends data. Adds to existing data. Also creates the file if not exist.

**;** : seperate command

**\** : newline command without execute

echo "first\_name,last\_name,age" > test-gen.csv;

scripting syntax-

<https://linuxhint.com/bash-scripting-symbols/>

Python in linux !

#!/usr/bin/python3

import sys

def usage():

print("usage: " + sys.argv[0] + " " + sys.argv[1])

print("number of arguments =", len(sys.argv)-1)

num\_args = len(sys.argv) - 1

if num\_args == 1:

pass

else:

usage()

sys.exit(1)

#!/usr/bin/python3

import sys

f = open("read.txt", "w") # creates a file (w- over write)

f.write("Hey world! my name is roey and i love peanut butter")

f.close

f = open("read.txt", "r")

words = f.read().split() # reads the text and separates it into list of words

print(words)

f.close