```
calender:
---------
sudo apt update
sudo apt install ncal
cal or ncal - calender with highlighted date
ncal <month> <year> - ncal 01 2025 -> provides jan 2025 calender

man command :
-------------

manul page (man)

$ man ls

this provides the manual for how to use `ls`

Basic Command syntax:
---------------------

<command> <args>
echo        hello

for shell we provide command followed by arguments

echo hello [echo is command and hello is argument]

There are two types of Arguments
    * Positional arguments
    * Named arguments

Positional Arguments :
-----------------------
<command> <arg1> <arg2>
cp 1.txt 2.txt

here position becomes very critical

Named arguments:
----------------
<command> --<arg> <argvalue>

ping -c 4 google.com


* to see hidden files
ls -a
```

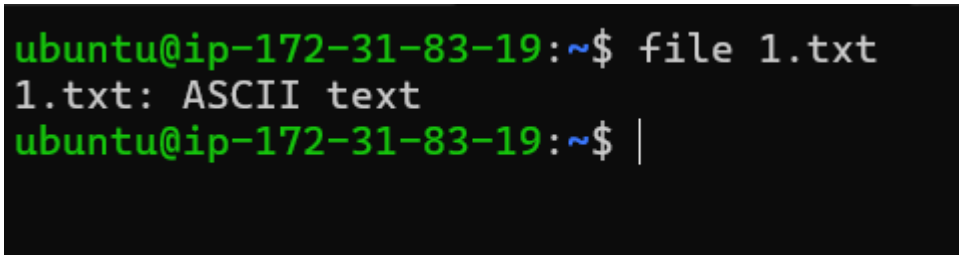
Linux Directory Hierarchy

- / => root directory

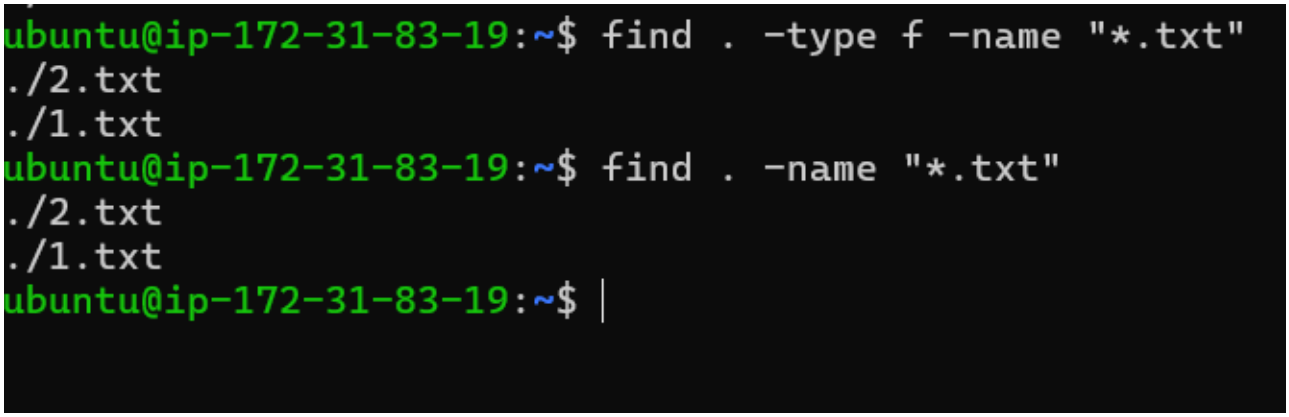
- /bin => Binaries and other executables
- /etc => system configuration files
- /home => home directory
- /opt => optional or third party softwares
- /tmp => Temporary spaces
- /usr => User related programs
- /var => variable data, log
- experiment with mkdir and rmdir.
- ls . [dot represents current directory]
- ls .. [one step back from current]
- ls ../lib [shows inside files of lib]

```
ubuntu@ip-172-31-83-19:~$ less 1.txt
ubuntu@ip-172-31-83-19:~$
ubuntu@ip-172-31-83-19:~$ touch 2.txt
ubuntu@ip-172-31-83-19:~$ diff -u 1.txt 2.txt
--- 1.txt       2024-08-01 02:03:21.100972929 +0000
+++ 2.txt       2024-08-01 02:06:05.571480447 +0000
@@ -1,33 +0,0 @@
-sadfasfasdfs
-asf
-as
-fas
-fs
-vas
-v
-a
-dvadsvdagwd
-g
-ad
-v
-sd
-v
-ad
-va
-dv
-a
-dv
-ad
-vbad
-gqdgadgadgagdas
-sf
-as
-
-as
-dg
-sag
-as
-f
-s
-g
-as
ubuntu@ip-172-31-83-19:~$ cp 1.txt 2.txt
ubuntu@ip-172-31-83-19:~$ diff -u 1.txt  2.txt
ubuntu@ip-172-31-83-19:~$
```

- file command is used to know more about that file

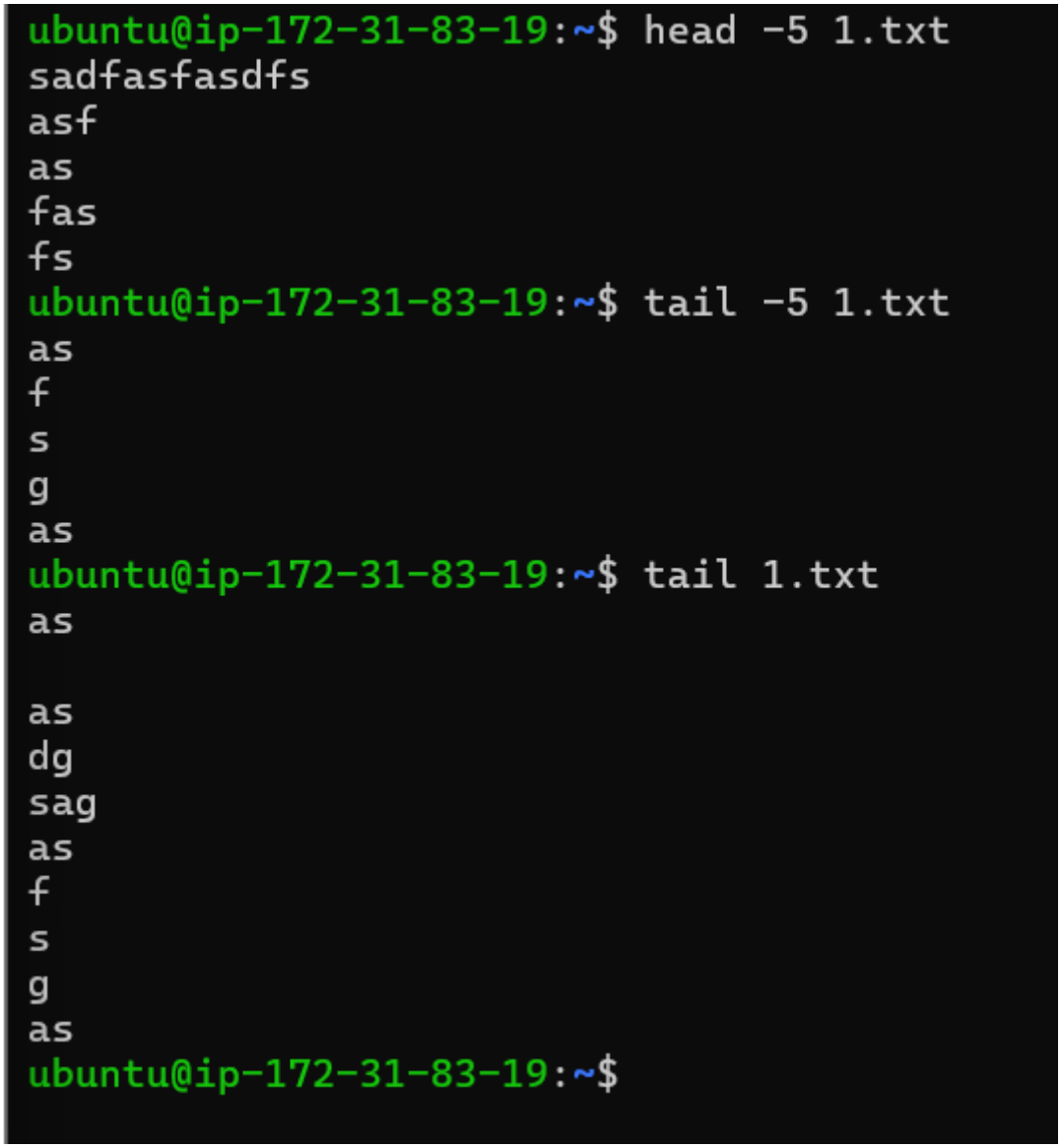


- find command



- head and tail commands
    - head command show fist n lines where n is an integer

- default tail command shows last 10 lines of file



Environmental and shell variables

- Shell variables : Shell can temporarily store variables called as shell variables
- once you logout and login back it cant remember previous values

```
<var-name>=<value>
Topic= linux
to access variable use $
echo $Topic
```

```
ubuntu@ip-172-31-83-19:~$ echo $Topic
linux
ubuntu@ip-172-31-83-19:~$ Topic=ubuntu
ubuntu@ip-172-31-83-19:~$ echo $topic

ubuntu@ip-172-31-83-19:~$ echo $Topic
ubuntu
ubuntu@ip-172-31-83-19:~$ exit
logout
Connection to 3.92.2.115 closed.
PS C:\Users\aravi> ssh ubuntu@3.92.2.115
Welcome to Ubuntu 24.04 LTS (GNU/Linux 6.8.0-1009-aws x86_64
)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Thu Aug  1 02:24:59 UTC 2024

  System load:  0.0               Processes:             115
  Usage of /:   26.0% of 6.71GB   Users logged in:       1
  Memory usage: 23%               IPv4 address for enX0: 172
.31.83.19
  Swap usage:   0%


Expanded Security Maintenance for Applications is not enable
d.

34 updates can be applied immediately.
18 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security update
s.
See https://ubuntu.com/esm or run: sudo pro status


Last login: Thu Aug  1 02:00:32 2024 from 103.197.112.85
ubuntu@ip-172-31-83-19:~$ echo $Topic

ubuntu@ip-172-31-83-19:~$
```

- to make it remember we write values in some files like /etc/environment, later even if you login back and use them

```
ubuntu@ip-172-31-83-19:~$ Topic=linux
ubuntu@ip-172-31-83-19:~$ echo $Topic
linux
ubuntu@ip-172-31-83-19:~$ sudo vi /etc/environment
ubuntu@ip-172-31-83-19:~$ source /etc/environment
ubuntu@ip-172-31-83-19:~$ exit
logout
Connection to 3.92.2.115 closed.
PS C:\Users\aravi> ssh ubuntu@3.92.2.115
Welcome to Ubuntu 24.04 LTS (GNU/Linux 6.8.0-1009-aws x86_64
)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Thu Aug  1 02:35:55 UTC 2024

  System load:  0.0               Processes:             113
  Usage of /:   26.0% of 6.71GB   Users logged in:       1
  Memory usage: 23%               IPv4 address for enX0: 172
.31.83.19
  Swap usage:   0%


Expanded Security Maintenance for Applications is not enable
d.

34 updates can be applied immediately.
18 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security update
s.
See https://ubuntu.com/esm or run: sudo pro status


Last login: Thu Aug  1 02:33:42 2024 from 103.197.112.85
ubuntu@ip-172-31-83-19:~$ echo $Topic
linux
ubuntu@ip-172-31-83-19:~$
```

CLASSESS

Class 1:

Directoroy Navigation

- . represents a present folder
- .. represents parent folder

- ~ Home folder
- linux contains folder and files
- when you get linux fully then it feels Everything in linux is a file.

Class 2:

Linux Directories

- Purpose of following directories
    - /etc
    - /bin
    - /home
    - /var
    - /tmp
- /etc :

Contains configuration files[password configuration,service files, user management, network configuration,package management(apt, dnf)]

- what is configuration ?
- configuration is change in settings of any program, for example : you can change chrome lightning mode from dark to colour or colour to dark this makes a chages in configuration files.
- In linux configuration files are present in /etc folder.
- simply it is settings for linux
- /bin :

Contains binary files

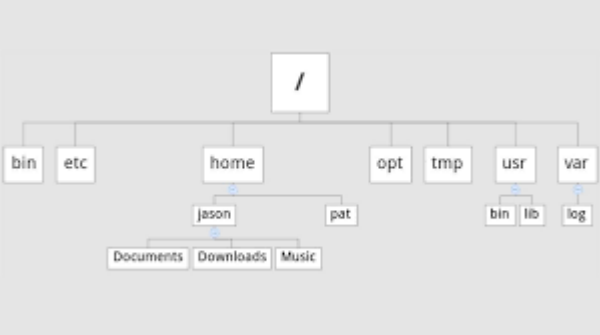
- binary files are files which are not human readable, they are machine readable that is OS. a file consists of instructions which an operating system can understand and whenever you executues it cpu and memory is allocating and program starts running
- executables
    - Binaris
        - all linux commands will exist here
        - when you do cat for binary files it displays in non-understandable languages
    - Scripts
        - when you cat for these files you can see the scripts
        - !#/bin/bash
- /home :

Home directory is used to store multiple users data, when multiple users use the machine they should have data under their names since all thse are under home directory , in windows it is c:/users folder.

- /var : Contains Variable data like logs which get changes all the time.

- tmp/ : Temporary files are stored here. Temporary files are files which are created for a short period of time and then deleted. For example,...when you download a file from the internet, it is first saved as a temporary file in the /tmp directory, and then moved to its final destination once the download is complete.

- tree structure of linux folders



Navigation and file system

- Absolute paths
- Relative paths
- Commands :
  - pwd - present directory
  - ls - list out files
  - cd - get into
- go to cd /var/log and do ls, now how to find whether the listed names are files or folders

```
-rw-rw-r--  1 root      utmp                4224 Aug 11 09:18 wtmp
ubuntu@ip-172-31-9-1:/var/log$ ls -al
total 588
drwxrwxr-x  11 root      syslog              4096 Aug 11 07:43 .
drwxr-xr-x  13 root      root                4096 Aug 11 07:43 ..
lrwxrwxrwx   1 root      root                  39 Jul  1 16:00 README -> ../..
/usr/share/doc/systemd/README.logs
-rw-r--r--   1 root      root                 444 Jul  1 16:03 alternatives.lo
g
drwx------   3 root      root                4096 Aug 11 07:43 amazon
-rw-r-----   1 root      adm                    0 Aug 11 07:43 apport.log
drwxr-xr-x   2 root      root                4096 Jul  1 16:06 apt
-rw-r-----   1 syslog    adm                41556 Aug 11 09:18 auth.log
-rw-rw----   1 root      utmp               31488 Aug 11 08:38 btmp
drwxr-x---   2 _chrony   _chrony             4096 Aug 11 07:43 chrony
-rw-r-----   1 root      adm                 4286 Aug 11 07:43 cloud-init-outp
```

- ls -al : list out files and folders with their permissions, ownership and size
- anything which is starting with d is directory.
- get into home directoy and find out hidden files
- anything which is starting with .txt is a hidden file.
- ls -a : list out all files and folders including hidden files

File Manupulation

- File Management
  - create :
    - touch => creates an empty file

    - edit
        - text editor
            - vim (learnig site openvim.com)
            - nano
    - delete
        - rm -r directoryname
        - rm -rf force deletion
        - rm -i file name -intractive deletion
        - rm *.txt -deletes all the files with extension .txt
- Folder Management
    - create
        - mkdir directoryname
        - mkdir f1 f2 d1 d2 -create multiple foders
    - edit
        - move
            - mv d1 d2 -folder d1 will move to folder d2.
    - delete
        - rmdir dirname -to delete an empty directory
        - rm -r directoryname
        - rm -rf force deletion
        - rm -i file name -intractive deletion
        - rm *.txt -deletes all the files with extension .txt

Users and Groups

Users

- Check Users in linux :
    - cat /etc/passwd
    - getent passwd
    - Check Single user:
        - sudo chage -l sai
    - Check whether account locked:(L defines locked)
        - sudo passwd -S sai
    - Check User exist or not
        - getent passwd | grep sai
    - list users above number 1000
        - getent passwd {1000..1010}
    - Check current user
        - `who` or `users`
- Create User :
    - sudo adduser aravindh
- Delete User :

    - sudo deluser aravindh
- Change password :
    - sudo passwd sai
- Set account expiery date
    - sudo chage -E 2025-05-30 sai
- Set account expiery to never
    - sudo chage -E -1 sai
- Lock User
    - sudo passwd -l sai
- Unlock User
    - sudo passwd -u sai
- change Username :
    - sudo usermod -l "saib" sai
- Add User to a group :
    - sudo usermod -aG group1 saib
- Remove User form a Group
    - sudo gpasswd -d saib group1
- This command sets:
    - Minimum days between password changes to 7 days.
    - Maximum days before the password expires to 90 days.
    - Warning days before expiration to 14 days.
    - Account expiration date to December 31, 2024.
    - `sudo chage -m 7 -M 90 -W 14 -E 2024-12-31 sai`

Groups

- Check groups in linux :
    - cat /etc/group
    - getent group
    - getent group | grep group1
    - getent group {1000..1010}
- To check groups assigned to him
    - `groups`
- To check groups assigned for specific user
    - groups saib
- To checks users identity

    - id sai
- To check id
    - id
- Create Newgroup :
    - sudo groupadd group1
- Delete Group :
    - sudo delgroup group1
- Display all users in specific group
    - getent group group1

VIM Editor :

- Basically VIM has two modes
    - insert mode
    - Normal mode
- insert mode allows you to write text same as text editor.
- Normal mode provides an efficient way to manipulate and navigate to text.
- At any time, you can see which mode you are in on the status bar which is located at the top of the editor.
- To change between modes, use `Esc` for normal mode and `i` for insert mode

Cursor movement :

- h - moves left similar to left navigation key <.
- l - moves right similar to right navigation key >.
- k - move top
- j - move down

Shell and Terminal

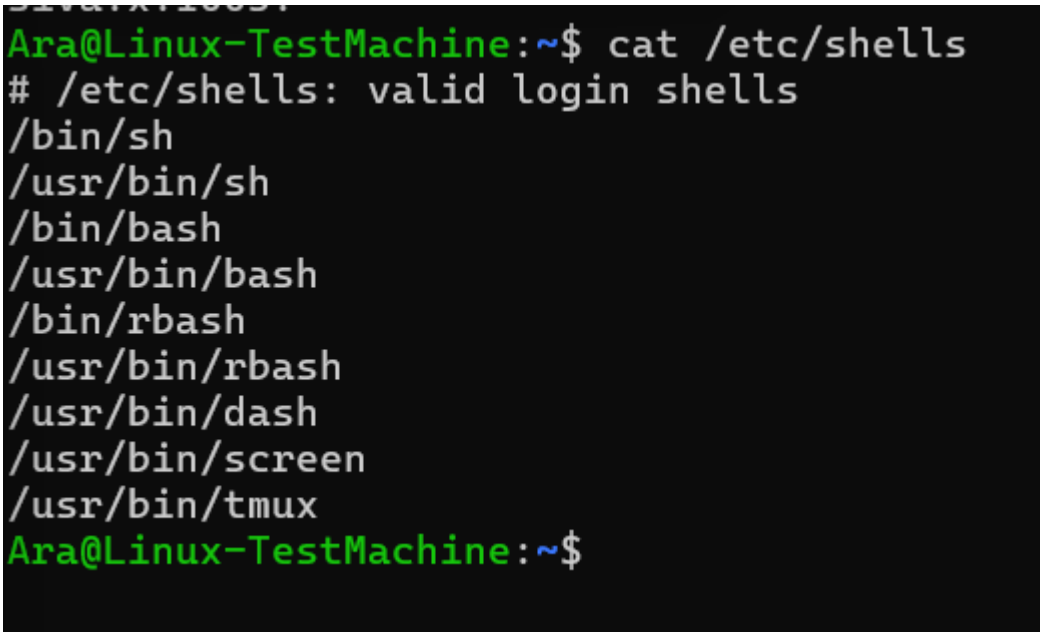
Termial :

- Terminal is a software that allows you to type commands ex: git,bash which are installed on your system

Shell :

- the commands which you wrote in the terminal understands by Shell.
- the software which understands the commands is called shell. windows : * DOS * PowerShell Linux : * bash * flash ..etc
- how to list all the shells

```
cat /etc/shells
```



- in Linux there are two types of users
    - Sytem Users
        - this is crerated to run some applications/services in linux
    - Users
- How to tell whether a user is System user / User.
    - when you see all the users list, where the users doesn't have 'nologin' at last is a system user
    - generally users will be associated with shell and system user will not be associated with shells
- On all linux machines we have `root` User, root user will have full control over machine
- how to change to root user

```
sudo -i
```

- to know who is current user

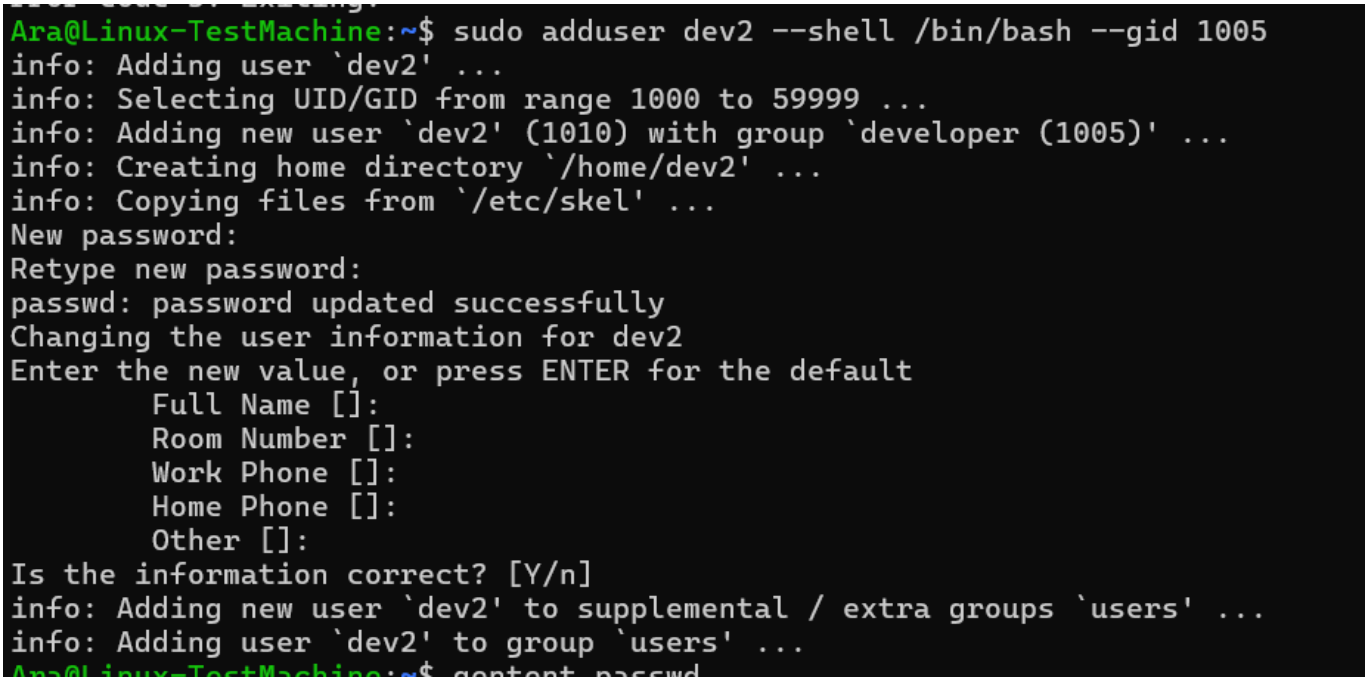
```
whoami
```

- users can be associated with the groups
- Lets create a Group
    - developer
    - tester
    - devops
- every user has a unique UID and every group will have unique GID

```
Username:password:UID:GID:GECOS:home_directory:login_shell
```

- the users will be listed in above format
- here when you create a user UID will be automatically created, and GID(group id) if you have not assigned to any group it will assign it will create a new group with username ang brings its id.
- if you assign any group, it shows respective group id.
- lets create a 3 Users
    - dev1
        - group : developer
        - shell : /bin/bash
        - home_directory : /home/dev1
    - test1
        - group : developer
        - shell : /bin/bash
        - home_directory : /home/dev1
    - devops1
        - group : developer
        - shell : /bin/bash
        - home_directory : /home/dev1

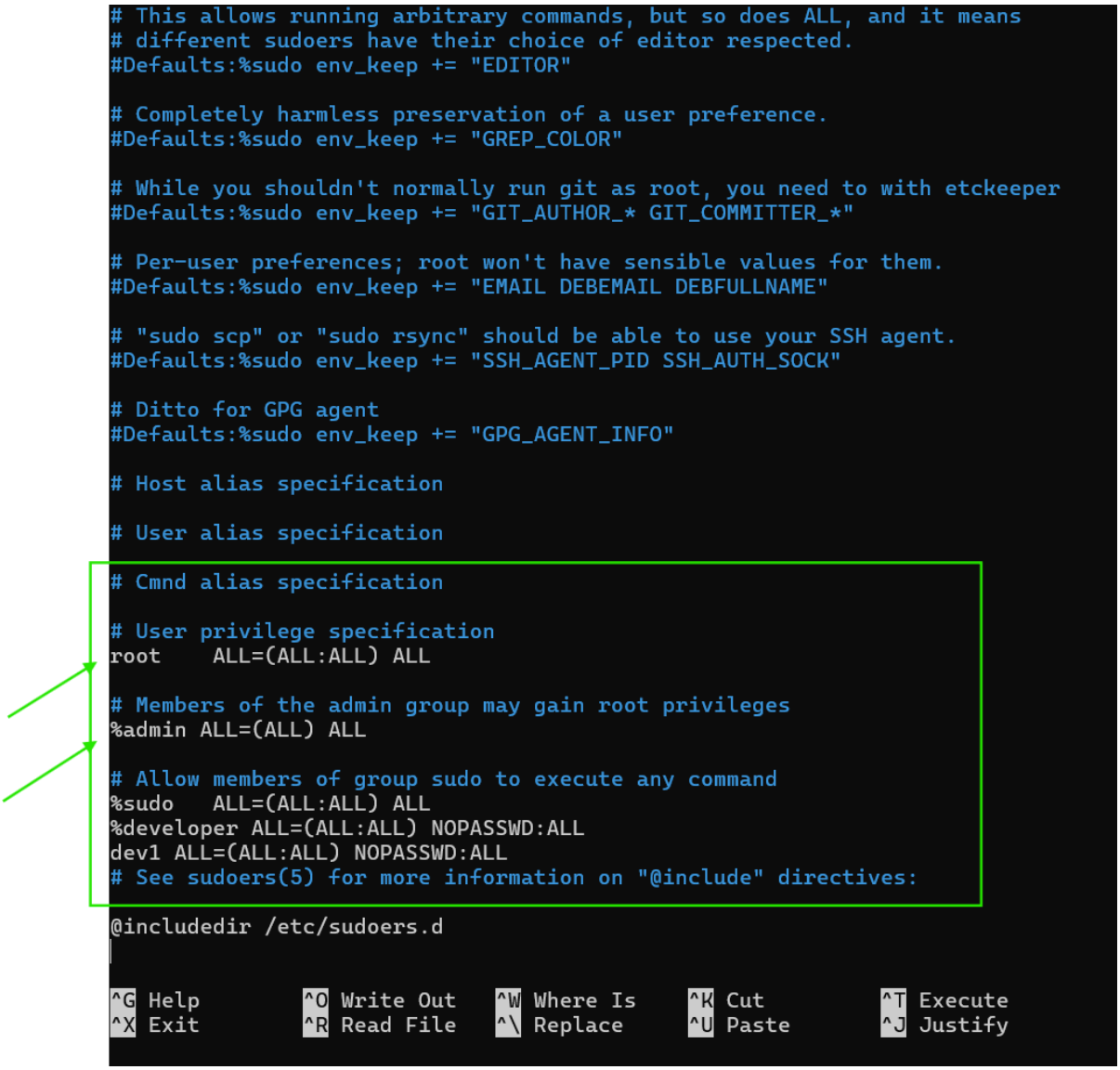
```
sudo adduser dev2 --shell /bin/bash --gid 1005
```




```
sudo useradd -s /bin/bash -g 1006 -m test1
```

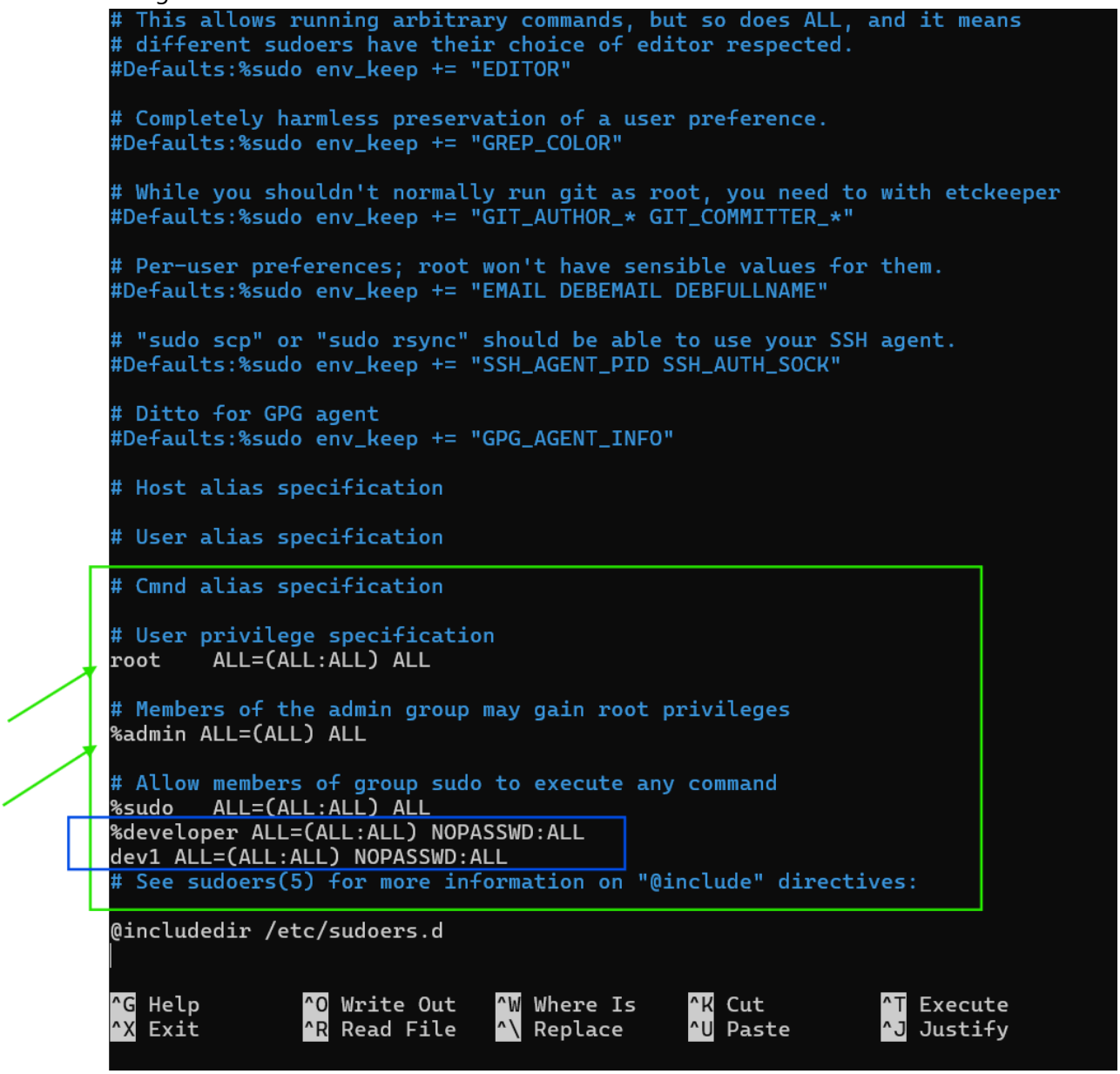
```
Ara@Linux-TestMachine:~$ sudo useradd -s /bin/bash -g 1006 -m test2
Ara@Linux-TestMachine:~$ getent passwd
```

- adding users to `sudoers` group
    - visudo
    - add to sudo group `usermod -aG sudo dev1`
- visudo :
    - open `sudo visuo`



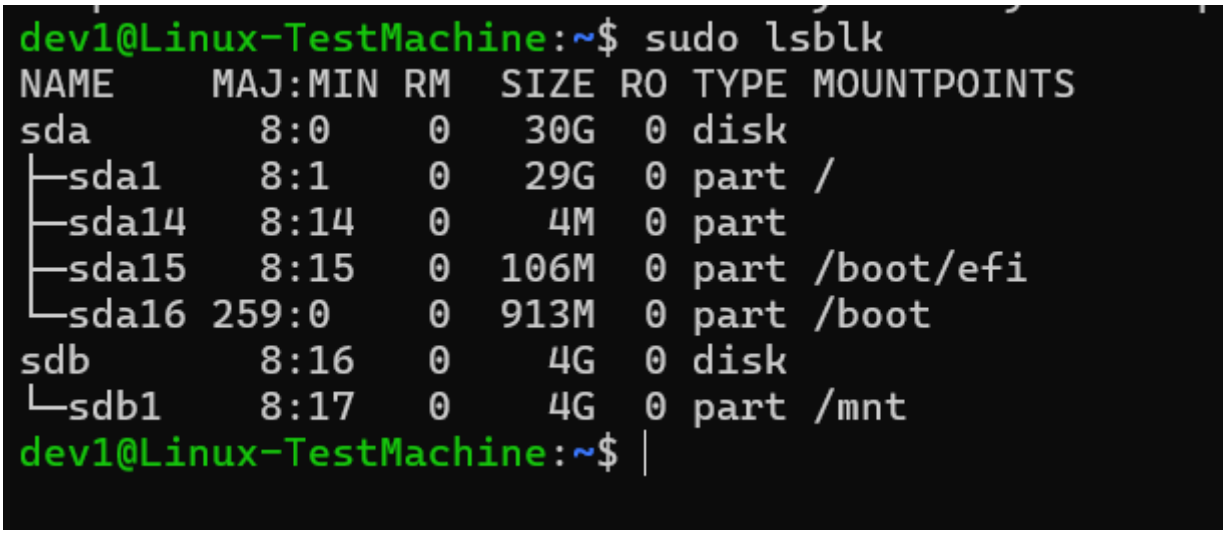
- if a line starts with `%` it is a group and if it starts normally it is a user.

- see above image



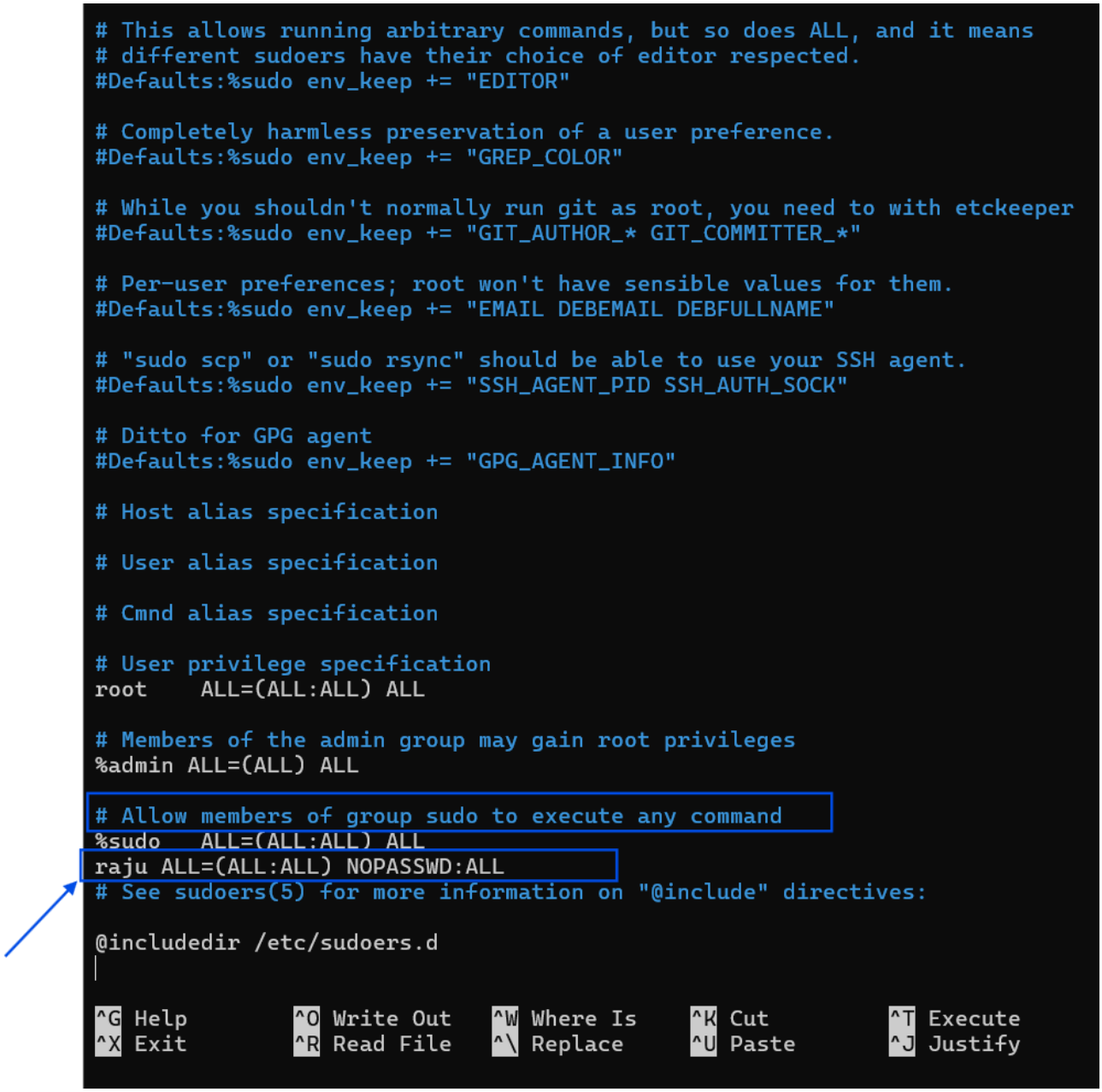
- see above image the blue colurd box content is written for not to ask password for group %developer and user dev1

to check all disks in linux machine



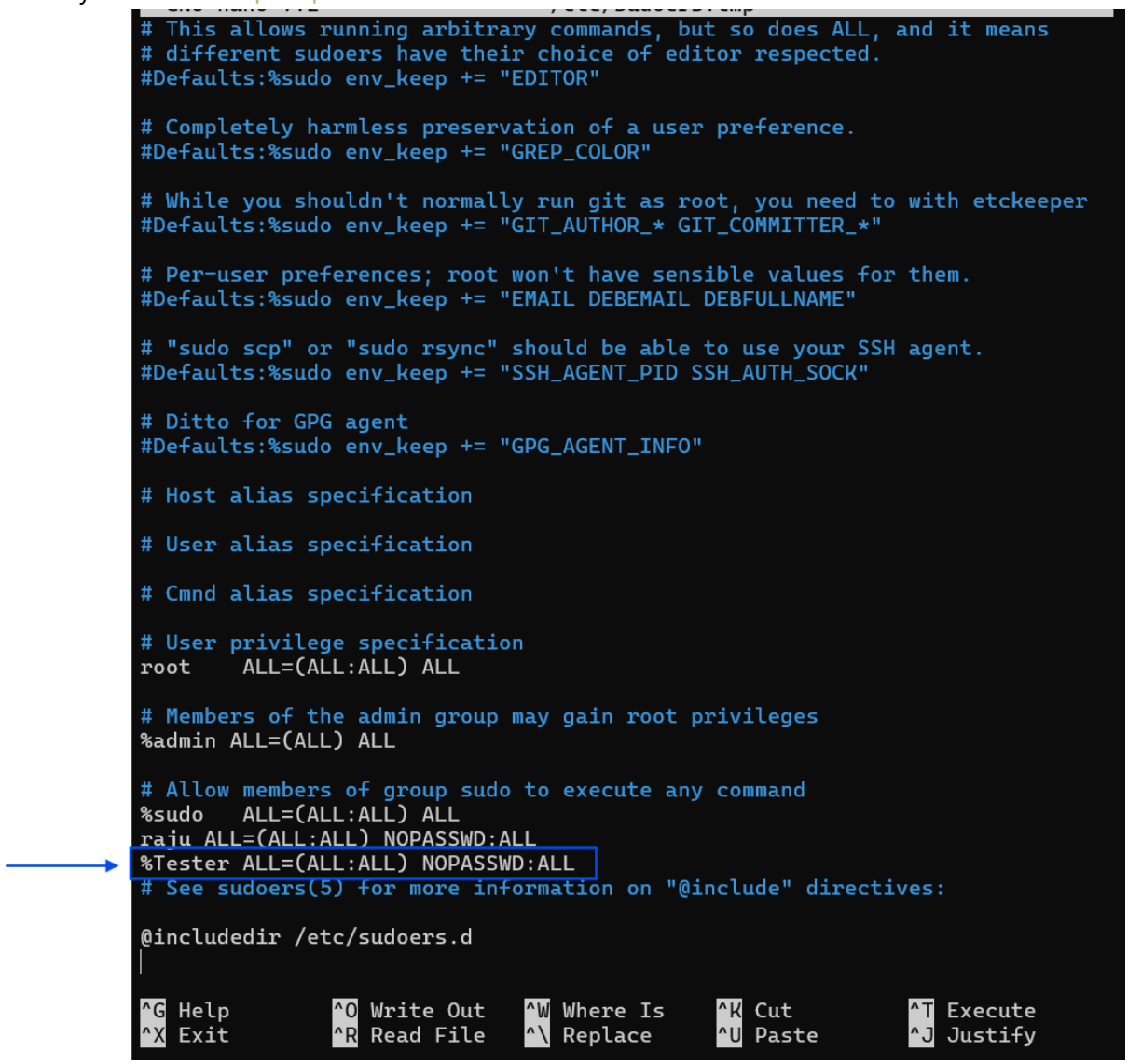
sudo visudo[second way of providing sudo permissions to user]

- if you create new user and add it in this file, he will get full permissions as `sudoers`

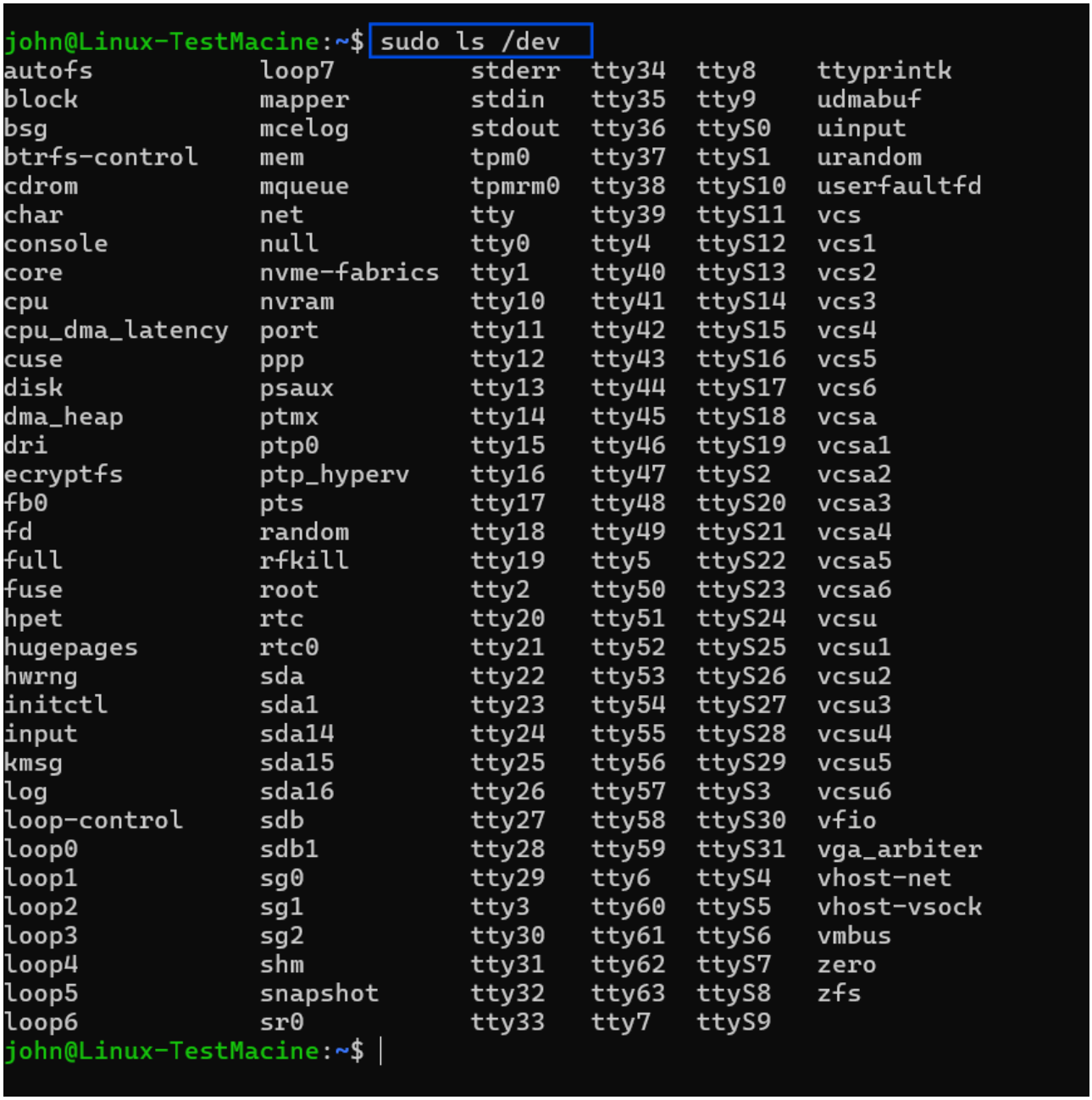


- if you create a new user, new group and add user to new group. then if you add this new group to this file the group will get `sudoers` permissions. ex: * create a new user john
    - create a new group Tester
    - add john to Tester group
    - open `sudo visudo` and add group with all permissions
        - `%Tester ALL(ALL:ALL) NOPASSWD:ALL`
    - save and exit.
    - now switch user `su john` using john

- now try to do `sudo apt update` it will work.



- this commad shows all the devices connected to the linux machine `sudo ls /dev`



FILE PERMISSIONS

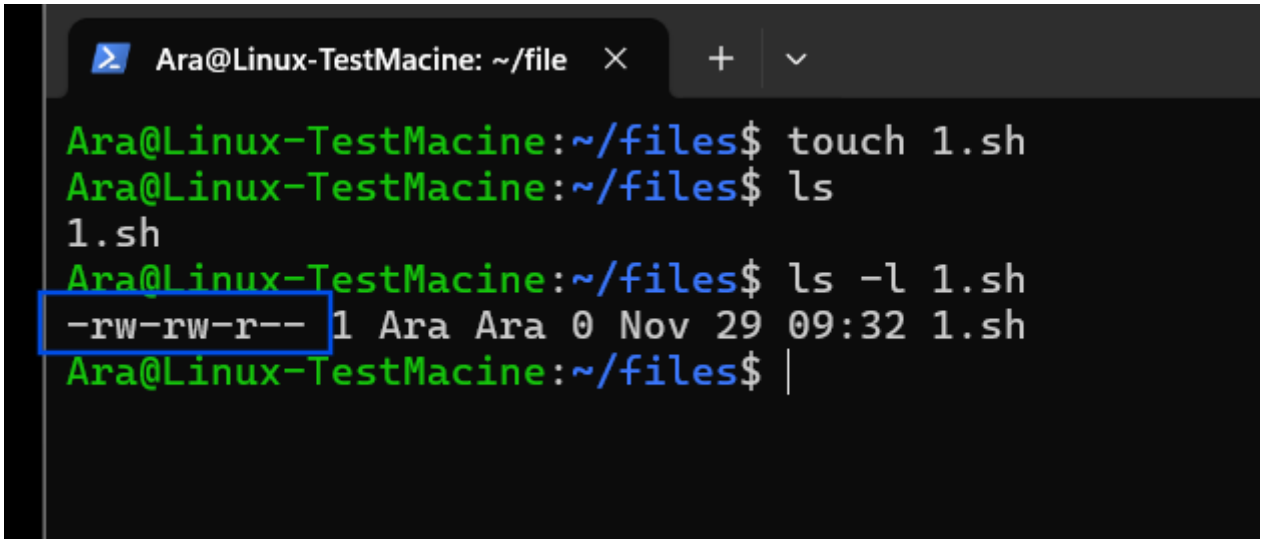
- `chmod` command is used to change file permissions
- `chown` command is used to change file ownership
- `chgrp` command is used to change file group ownership

chmod

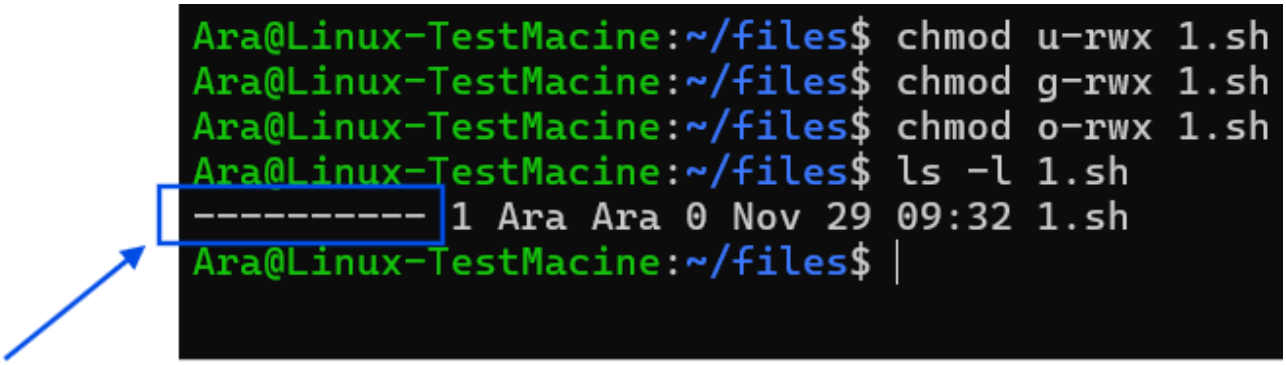
Practical :

- Create a new file `touch 1.sh`

- do `ls -l 1.sh`

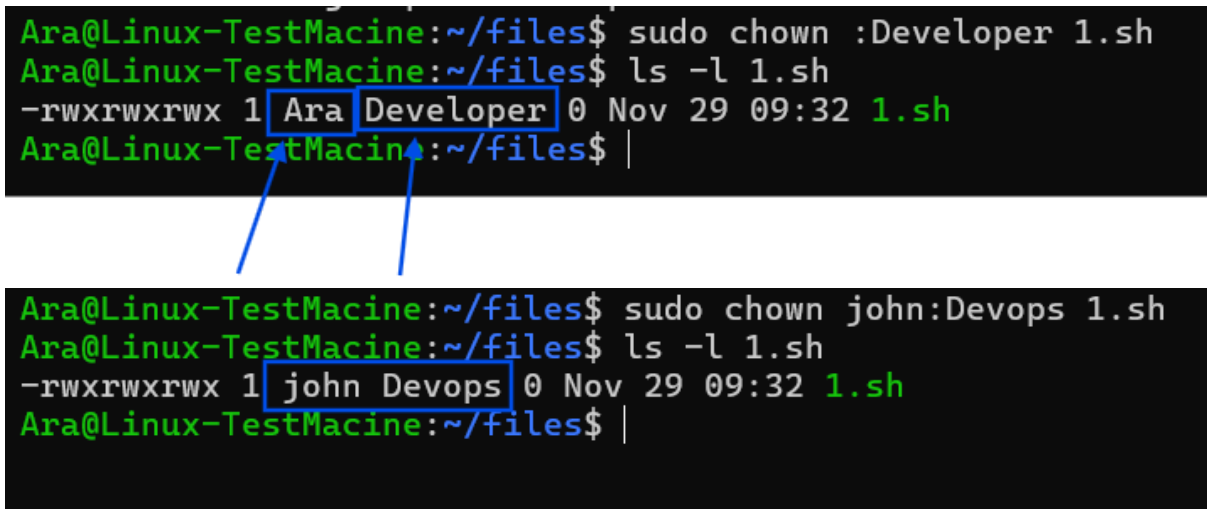


- removing all permissions



- here we have three diffrent access `owner group other`.
- Owner = u
- Group = g
- Other = o
- the command comes as below
    - `chmod u+r 1.sh` - single permission at a time
    - `chmod u+rw 1.sh` - double pemission at a time
    - `chmod u+rwx 1.sh` - triple permission at a time
    - `chmod g+r 1.sh`
    - `chmod g+rw 1.sh`
    - `chmod g+rwx 1.sh`
    - `chmod o+r 1.sh`
    - `chmod o+rw 1.sh`

- `chmod o+rwx 1.sh`



Understanding Linux Software Installations

Installing software in Linux involves ensuring executables are present and configured correctly. You can achieve this using:

1. **Source Code**
2. **Precompiled Binaries**
3. **Packages** (via package managers)

---

1. Installing Software From Source Code

This method involves downloading, building, and installing software manually.

Example: Installing `htop` from Source

1. **Download the source code**:

```
wget https://github.com/htop-dev/htop/archive/refs/heads/main.zip
unzip main.zip
cd htop-main
```

2. **Build and install**:

```
./autogen.sh  # Prepare the build system
./configure    # Configure the installation
make           # Build the software
sudo make install  # Install it system-wide
```

3. **Verify Installation**:
   Run `htop` in the terminal:

```
htop
```

Requirements:

- Install build tools:

```
sudo apt install build-essential  # For Debian-based
sudo dnf groupinstall "Development Tools"  # For RedHat-based
```

---

2. Installing Precompiled Binaries

This method uses pre-built executables directly without building from source.

Example: Installing Apache Maven

1. **Download the binary**:

```
wget https://downloads.apache.org/maven/maven-3/3.9.5/binaries/apache-maven-3.9.5-bin.tar.gz
```

2. **Extract the archive**:

```
tar -xvzf apache-maven-3.9.5-bin.tar.gz
sudo mv apache-maven-3.9.5 /opt/maven
```

3. **Set up environment variables**: Edit `.bashrc` or `.zshrc`:

```
export M2_HOME=/opt/maven
export PATH=$M2_HOME/bin:$PATH
```

Reload the shell:

```
source ~/.bashrc
```

4. **Verify installation**:

```
mvn -version
```

3. Installing Software Using Packages

Packages are pre-built and are managed by package managers for easier installations.

Package Managers Overview

Package Manager	Command	Example
APT (Debian)	`apt install`	`sudo apt install vim`
DNF (RedHat)	`dnf install`	`sudo dnf install httpd`
RPM (RedHat)	`rpm -i`	`sudo rpm -ivh example.rpm`
DPKG (Debian)	`dpkg -i`	`sudo dpkg -i example.deb`

Examples for Real-World Scenarios

Debian Package Installation

Example: Installing Google Chrome

1. **Download the `.deb` file**:

```
wget https://dl.google.com/linux/direct/google-chrome-
stable_current_amd64.deb
```

2. **Install the package**:

```
sudo dpkg -i google-chrome-stable_current_amd64.deb
```

3. **Fix dependencies (if any)**:

```
sudo apt install -f
```

4. **Run the application**:

```
google-chrome
```

RedHat Package Installation

Example: Installing Docker

1. **Add Docker repository**:

```
sudo dnf config-manager --add-
repo=https://download.docker.com/linux/centos/docker-ce.repo
```

2. **Install Docker**:

```
sudo dnf install docker-ce docker-ce-cli containerd.io
```

3. **Start the Docker service**:

```
sudo systemctl start docker
sudo systemctl enable docker
```

4. **Verify installation**:

```
docker --version
```

Application Types

1. Standalone Applications

Applications that can be run directly.
Example: Running Visual Studio Code portable version:

1. Download the portable `.tar.gz` file from the Visual Studio Code website.
2. Extract it:

```
tar -xvzf code-stable-x64.tar.gz
cd VSCode-linux-x64
./code
```

---

2. Service/Daemon Applications

Applications that run in the background and are controlled via `systemctl`.

Example: Installing and Configuring Nginx

1. **Install Nginx**:

```
sudo apt install nginx  # Debian-based
sudo dnf install nginx  # RedHat-based
```

2. **Start the Nginx service**:

```
sudo systemctl start nginx
```

3. **Enable the service to start at boot**:

```
sudo systemctl enable nginx
```

4. **Check the status**:

```
sudo systemctl status nginx
```

5. **Access Nginx in a browser**: Open `http://<your-ip-address>` in your browser.

Service File Configuration

For daemon-based applications, you may need to edit service files stored in:

- `/etc/systemd/system/` or `/lib/systemd/system/`

Example: Customizing Nginx Service: Edit the service file:

```
sudo nano /lib/systemd/system/nginx.service
```

Reload systemd to apply changes:

```
sudo systemctl daemon-reload
sudo systemctl restart nginx
```

General Tips

- Always update the package manager:

```
sudo apt update && sudo apt upgrade  # Debian-based
sudo dnf update                      # RedHat-based
```

- Remove unused packages:

```
sudo apt autoremove  # Debian-based
sudo dnf autoremove  # RedHat-based
```

- For GUI applications, ensure a desktop environment is installed.

By understanding these methods and examples, you can handle almost any Linux software installation. Let me know if you need help with specific software!

MVN installation

```
Aravindh@VM1:~$ mvn --version
Command 'mvn' not found, but can be installed with:
sudo apt install maven
Aravindh@VM1:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr
/local/games:/snap/bin
Aravindh@VM1:~$ sudo vi $PATH
Aravindh@VM1:~$ ls
Aravindh@VM1:~$ cd /tmp/
Aravindh@VM1:/tmp$ sudo mv apache-maven-3.9.9 /opt/
Aravindh@VM1:/tmp$ cd /opt/
Aravindh@VM1:/opt$ ls
apache-maven-3.9.9
Aravindh@VM1:/opt$ cd apache-maven-3.9.9/bin/
Aravindh@VM1:/opt/apache-maven-3.9.9/bin$ ls
m2.conf  mvn  mvn.cmd  mvnDebug  mvnDebug.cmd  mvnyjp
Aravindh@VM1:/opt/apache-maven-3.9.9/bin$ sudo vi /etc/environment
Aravindh@VM1:/opt/apache-maven-3.9.9/bin$ source /etc/environment
Aravindh@VM1:/opt/apache-maven-3.9.9/bin$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr
/local/games:/snap/bin:/opt/apache-maven-3.9.9/bin
Aravindh@VM1:/opt/apache-maven-3.9.9/bin$ cd ~
Aravindh@VM1:~$ mvn --version
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcdc97d260186937)
Maven home: /opt/apache-maven-3.9.9
Java version: 17.0.13, vendor: Ubuntu, runtime: /usr/lib/jvm/java-17-openjdk
-amd64
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "6.5.0-1025-azure", arch: "amd64", family: "unix"
Aravindh@VM1:~$
```

outputs

In Linux, `stdin`, `stdout`, and `stderr` are the three standard streams used for input, output, and error handling, respectively. Here's a breakdown of their use cases with real-time examples:

---

1. Standard Input (`stdin`)

- **Purpose:** Used to take input from the user or another process.
- **Default source:** Keyboard.

Use Cases:

- Accepting user input in a script or program.
- Redirecting input from a file or another command.

Examples:

Interactive Input:

```
read name
echo "Hello, $name"
```

- Prompts the user for their name and then greets them.

Redirect Input from a File:

```
cat < input.txt
```

- Reads the contents of `input.txt` and outputs it to the terminal.

Pipe Input to Another Command:

```
echo "Linux is great!" | wc -w
```

- Outputs `3` (word count of the input).

---

2. Standard Output (`stdout`)

- **Purpose:** Used to display output to the terminal or another process.
- **Default target:** Terminal.

Use Cases:

- Printing program results.
- Redirecting output to a file or another command.

Examples:

Display Output to Terminal:

```
echo "This is standard output"
```

Redirect Output to a File:

```
ls > file_list.txt
```

- Saves the list of files in the current directory to `file_list.txt`.

Pipe Output to Another Command:

```
ls | grep "test"
```

- Lists files with names containing "test".

---

3. Standard Error (`stderr`)

- **Purpose:** Used to display error messages.
- **Default target:** Terminal.

Use Cases:

- Displaying error messages separately from normal output.
- Redirecting error messages to a file or another stream.

Examples:

Display Error to Terminal:

```
ls nonexistentfile
```

- Outputs an error like `ls: cannot access 'nonexistentfile': No such file or directory`.

Redirect Error to a File:

```
ls nonexistentfile 2> error_log.txt
```

- Saves the error message to `error_log.txt`.

Suppress Error Messages:

```
ls nonexistentfile 2>/dev/null
```

- Silently ignores errors by redirecting them to `/dev/null`.

Combine `stdout` and `stderr`:

```
ls existingfile nonexistentfile > all_output.txt 2>&1
```

- Merges both output and error streams into `all_output.txt`.

Advanced Use Cases

1. Using `stdin` and `stdout` Together:

```
cat > output.txt
```

- Takes user input (via `stdin`) and writes it to `output.txt` (via `stdout`).

2. Using `stdout` and `stderr` Together:

```
find / -name "somefile" > result.txt 2> error_log.txt
```

- Sends normal output to `result.txt` and errors to `error_log.txt`.

3. Pipeline with All Streams:

```
find / -name "somefile" 2> error_log.txt | grep "test" > final_output.txt
```

- Finds files, filters the results with `grep`, and handles errors separately.

4. Silent Mode for Scripts:

```
./script.sh > /dev/null 2>&1
```

- Suppresses both output and error messages by redirecting them to `/dev/null`.

These streams offer flexibility for processing input, output, and errors efficiently in Linux environments.

TEE Commands

The `tee` command in Linux reads from `stdin` and writes to both `stdout` and one or more files. It is commonly used for logging, debugging, and processing streams. Below are all the usage cases of `tee` with examples.

---

1. Basic Usage

Writes the input to a file and displays it on the terminal (stdout).

Example:

```
echo "Hello, Linux!" | tee file.txt
```

- Displays `Hello, Linux!` on the terminal and writes it to `file.txt`.

---

2. Append to a File

Use the `-a` or `--append` option to append data to an existing file instead of overwriting it.

Example:

```
echo "Appended line" | tee -a file.txt
```

- Appends `Appended line` to `file.txt`.

---

3. Multiple File Outputs

Write to multiple files simultaneously.

Example:

```
echo "Multi-file output" | tee file1.txt file2.txt
```

- Writes `Multi-file output` to both `file1.txt` and `file2.txt` and displays it on the terminal.

---

4. Combining with Pipes

Capture and process output in a pipeline while saving it to a file.

Example:

```
ls | tee filelist.txt | grep "test"
```

- Saves the output of `ls` to `filelist.txt` and filters lines containing "test" for display.

---

5. Suppress Output to Terminal

Redirect `stdout` to `/dev/null` to write to files only.

Example:

```
echo "Silent file write" | tee file.txt > /dev/null
```

- Writes `Silent file write` to `file.txt` without displaying it on the terminal.

---

6. Debugging and Logging

Capture command output in real-time while viewing it on the terminal.

Example:

```
ping -c 5 google.com | tee ping.log
```

- Saves the output of `ping` to `ping.log` and shows it on the terminal simultaneously.

---

7. Combining with Sudo

Write to files requiring elevated privileges.

Example:

```
echo "System-wide config" | sudo tee /etc/system.conf
```

- Writes `System-wide config` to `/etc/system.conf` with root permissions.

---

8. Use with Filters

Combine with other utilities for data processing.

Example:

```
cat file.txt | tee >(grep "error" > errors.log) >(grep "info" > info.log)
```

- Splits the input into multiple files based on content filtering.

---

9. Avoid Overwriting Files

Prevent overwriting sensitive files by using the `-i` (interactive) option.

Example:

```
echo "Overwrite check" | tee -i file.txt
```

- Prompts for confirmation if `file.txt` already exists.

---

10. Combine `tee` with Other Streams

Save both `stdout` and `stderr` to a file.

Example:

```
find / -name "somefile" 2>&1 | tee output.log
```

- Captures both the normal output and errors into `output.log`.

---

11. Measure and Save Execution Time

Log the execution time of a command while displaying it.

Example:

```
(time ls) 2>&1 | tee time.log
```

- Saves the command's output and its execution time to `time.log`.

---

12. Debug Shell Scripts

Log the output of a script for debugging purposes.

Example:

```

```

```
./script.sh | tee debug.log
```

- Captures the script's output in real-time and saves it to `debug.log`.

---

13. Create Backups While Writing

Write to a file and simultaneously create a backup.

Example:

```
echo "Critical data" | tee file.txt >(cp /dev/stdin backup.txt)
```

- Writes `Critical data` to both `file.txt` and `backup.txt`.

---

14. Chain Multiple Commands

Use `tee` in command chains to capture intermediate outputs.

Example:

```
ls | tee intermediate.log | sort | tee sorted.log
```

- Saves unsorted and sorted outputs into `intermediate.log` and `sorted.log`.

The `tee` command is a versatile tool for stream duplication and logging, making it invaluable for Linux users and system administrators.

REGULAR EXPRESSIONS

Complete Guide on Regular Expressions (Regex)

Regular expressions (regex) are powerful tools used for pattern matching and manipulation of strings in text processing tasks. Regex is supported in many programming languages and tools like Python, Java, JavaScript, sed, grep, awk, and more.

Components of Regex

1. Literals

Literal characters match themselves exactly. **Example:**

- Pattern: `cat`

- Matches: "cat" in "A cat on the mat."

---

2. Meta-characters

These are special symbols with predefined meanings.

Meta-character	Description	Example
`.`	Matches any single character	`c.t` matches "cat", "cot", etc.
`^`	Matches start of a line or string	`^cat` matches "cat" at the start
`$`	Matches end of a line or string	`cat$` matches "cat" at the end
`*`	Matches 0 or more of the preceding	`ca*t` matches "ct", "cat", etc.
`+`	Matches 1 or more of the preceding	`ca+t` matches "cat", not "ct"
`?`	Matches 0 or 1 of the preceding	`ca?t` matches "ct" or "cat"
`{n,m}`	Matches between n and m repetitions	`a{2,4}` matches "aa", "aaa", etc.

---

3. Character Classes

Define sets of characters to match.

Class	Description	Example
`[abc]`	Matches any character in brackets	`gr[ae]y` matches "gray", "grey"
`[^abc]`	Matches any character not in brackets	`[^aeiou]` matches consonants
`[a-z]`	Matches any character in the range a to z	`[a-z]` matches "a", "b", etc.
`\d`	Matches any digit (same as `[0-9]`)	`\d{3}` matches "123"
`\w`	Matches word characters (alphanumeric + _)	`\w+` matches "word123"
`\s`	Matches whitespace characters (spaces, tabs)	`\s+` matches spaces

---

4. Anchors

Anchors do not match characters but positions in the text.

Anchor	Description	Example
`^`	Matches start of a line	`^Hello` matches "Hello" at start
`$`	Matches end of a line	`world$` matches "world" at end
`\b`	Matches word boundary	`\bcat\b` matches "cat"
`\B`	Matches non-word boundary	`\Bcat` matches "concat"

---

5. Escape Sequences

Escape meta-characters with `\` if you want to use them literally. **Example:**

- Pattern: `\.` matches a literal period (`.`).

---

6. Groups and Capturing

Groups capture substrings for reuse.

Feature	Description	Example
`(abc)`	Captures "abc" for backreference	`(ab)c` matches "abc"
`(?:abc)`	Non-capturing group	`(?:ab)c` matches "abc" but not capture "ab"
`\1`	Refers to the first captured group	`(.)\1` matches "aa", "bb", etc.

---

7. Lookarounds

Matches patterns based on conditions without consuming text.

Lookaround	Description	Example
`(?=abc)`	Positive lookahead (followed by "abc")	`a(?=b)` matches "a" in "ab"
`(?!abc)`	Negative lookahead (not followed by "abc")	`a(?!b)` matches "a" in "ac"
`(?<=abc)`	Positive lookbehind (preceded by "abc")	`(?<=a)b` matches "b" in "ab"
`(?<!abc)`	Negative lookbehind (not preceded by "abc")	`(?<!a)b` matches "b" in "cb"

Common Use Cases

1. Validating Input

Pattern: `^\d{10}$`

- Matches a 10-digit phone number.
- **Example:** `1234567890` is valid, `12345` is not.

2. Extracting Email Addresses

Pattern: `[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}`

- Matches valid email addresses.
- **Example:** Extracts "example@gmail.com" from "Contact: example@gmail.com".

3. Replacing Text

Use regex in tools like `sed` or programming languages like Python. **Example:**

```
echo "apple banana orange" | sed 's/banana/mango/'
```

- Output: `apple mango orange`.

4. Extracting Log Information

Pattern: `ERROR: (.+)`

- Extracts error messages from logs.
- **Example:** "ERROR: Disk not found" extracts "Disk not found".

5. Splitting Strings

Pattern: `,`

- Splits CSV strings.
- **Example:** `"name,age,location"` → `["name", "age", "location"]`.

6. Filtering Files

Command:

```
ls | grep '\.txt$'
```

- Lists all `.txt` files.

7. Password Validation

Pattern: `^(?=.*[A-Z])(?=.*\d)[A-Za-z\d]{8,}$`

- Ensures at least one uppercase letter, one digit, and 8+ characters.

8. Removing Extra Spaces

Pattern: `\s+` **Command:**

```
echo "Too    many spaces" | sed 's/\s\+/ /g'
```

- Output: `Too many spaces`.

Regex in Real-Time Tools

1. Programming (Python Example)

```
import re

text = "Email me at example@gmail.com or support@domain.com"
emails = re.findall(r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}', text)
print(emails)  # Output: ['example@gmail.com', 'support@domain.com']
```

2. Command-Line (grep)

```
grep -Eo '[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}' file.txt
```

- Extracts all email addresses from `file.txt`.

3. Text Editors

In VSCode or Sublime, regex-based find-and-replace can modify large files quickly.

Best Practices

1. **Escape meta-characters** if used literally.
2. Use **non-capturing groups** (`(?:...)`) when groups are not reused.
3. Test regex patterns with tools like Regex101.
4. Use **verbose mode** (e.g., `re.VERBOSE` in Python) for complex patterns.

Mastering regex unlocks incredible efficiency in text manipulation, validation, and processing across platforms and programming languages!