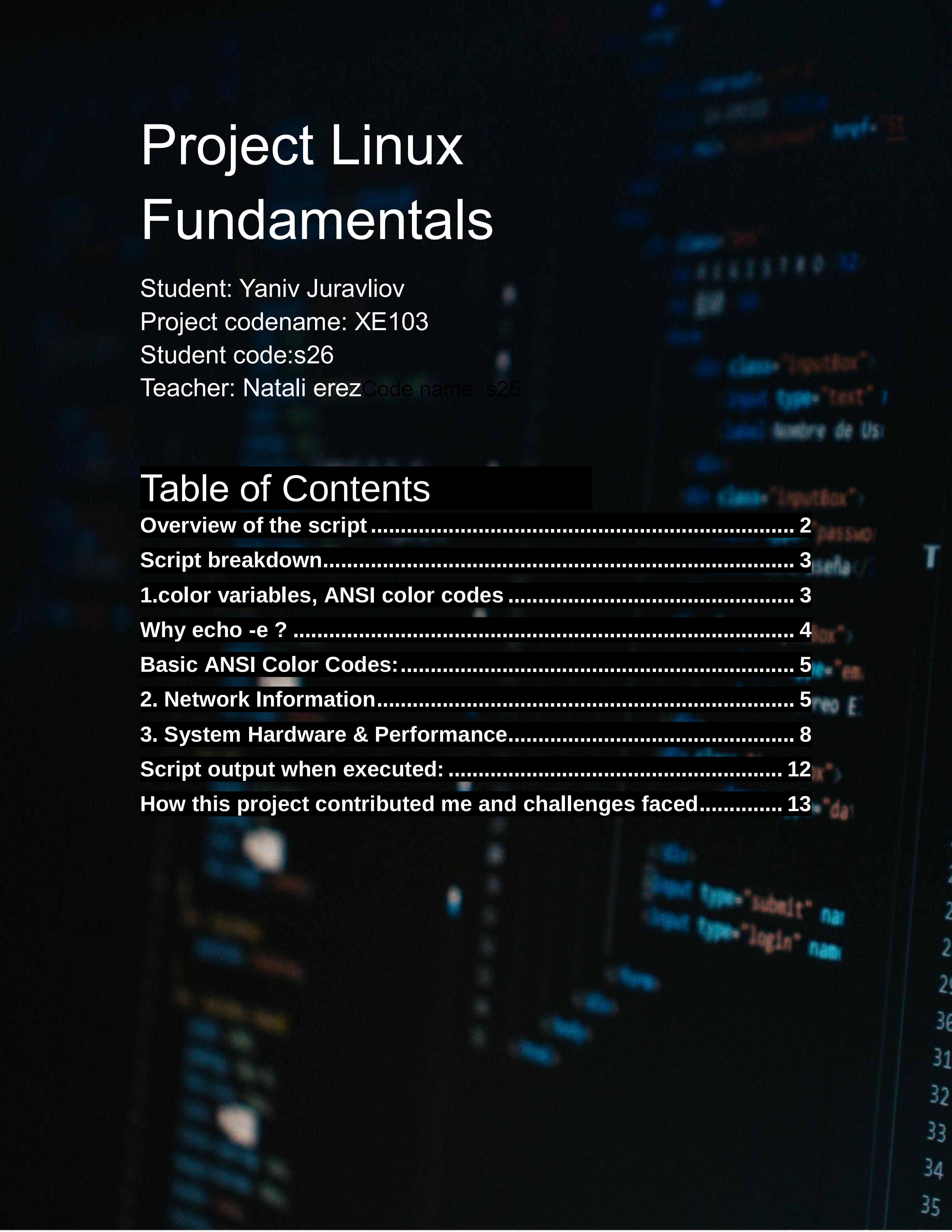


Project Linux Fundamentals

Student: Yaniv Juravliov
Project codename: XE103
Student code:s26
Teacher: Natali erezCode name: s26

Table of Contents

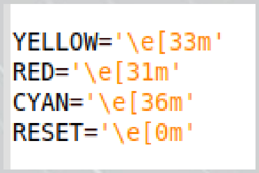
Overview of the script

The following Bash script is designed to **collect and display key system and network information** from a Linux machine in an organized and visually appealing way.
By using color formatting and built-in system commands, it provides a clear snapshot of the system's **network identity**, **hardware utilization**, **active processes**, and optionally the **largest files in the user's home directory**.

The script runs in two main sections, one focusing on **network details**, and the other on **system resource and hardware information,** making it useful for **administrators, analysts, or students** who need a quick overview of system performance and configuration.
It uses functions for modularity, ensures readability through color-coded output, and offers interactive input to display additional data when desired.

Script breakdown

1.color variables, ANSI color codes



```
YELLOW='\e[33m'
RED='\e[31m'
CYAN='\e[36m'
RESET='\e[0m'
```

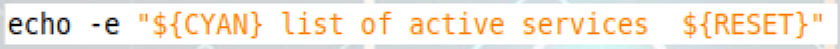
These ANSI escape codes define colors for terminal text:

- YELLOW, RED, CYAN are used to make sections visually distinct.
- RESET restores the default terminal color after each section.

This improves readability, especially when scanning through different types of information.

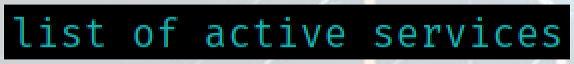
ANSI escape codes are used to control formatting, including color, in a Bash terminal. These codes are sequences of characters that begin with the escape character \e (or \033).

Example:



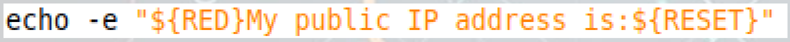
```
echo -e "${CYAN} list of active services  ${RESET}"
```

Result:



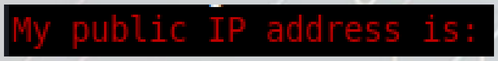
```
list of active services
```

Another example:



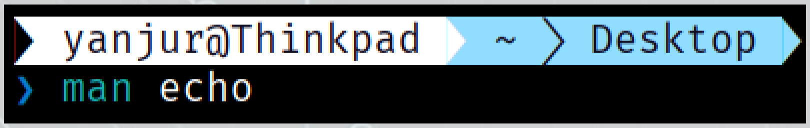
```
echo -e "${RED}My public IP address is:${RESET}"
```

Result:

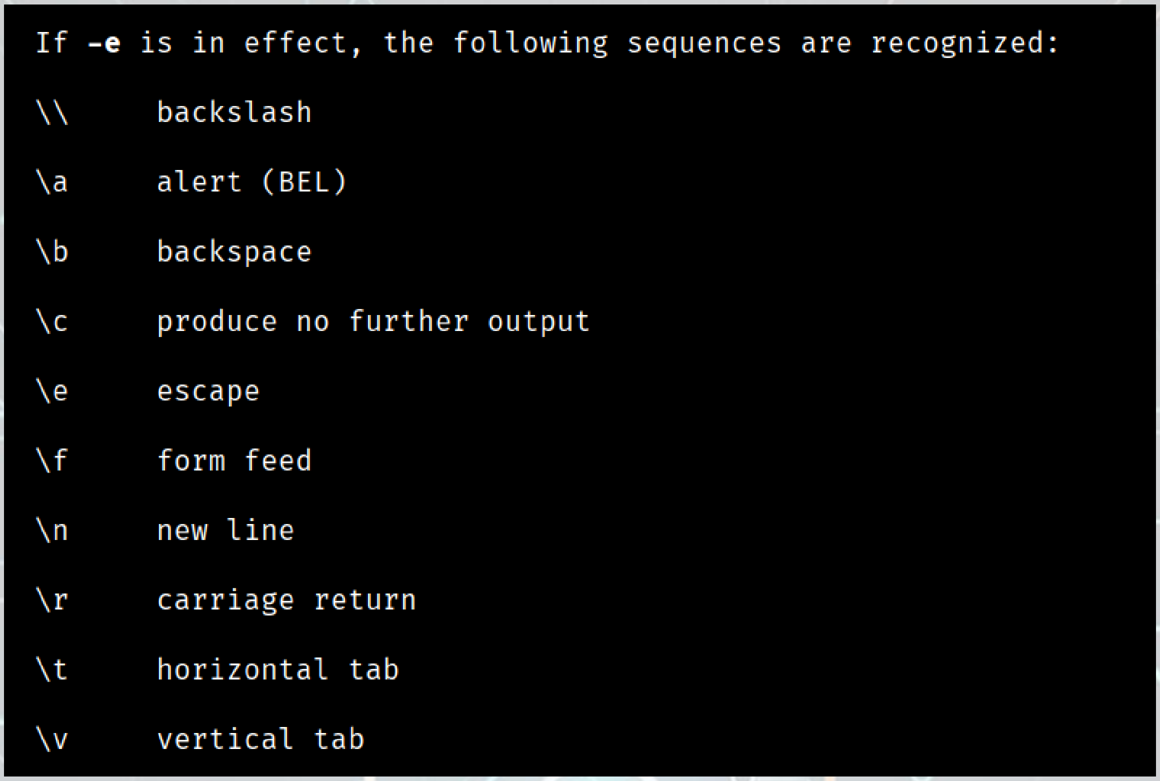


```
My public IP address is:
```

Why echo -e ?

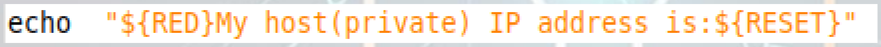


```
❯ yanjur@Thinkpad  ~  Desktop
❯ man echo
```

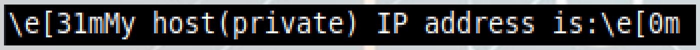


```
If -e is in effect, the following sequences are recognized:

\\      backslash

\a      alert (BEL)

\b      backspace

\c      produce no further output

\e      escape

\f      form feed

\n      new line

\r      carriage return

\t      horizontal tab

\v      vertical tab
```

Let’s try without -e



```
echo  "${RED}My host(private) IP address is:${RESET}"
```



```
\e[31mMy host(private) IP address is:\e[0m
```

So in order for “echo” to work properly we need to apply the **-e** flag that way it can recognize the backslash and apply the ANSI color which I assigned as a variable

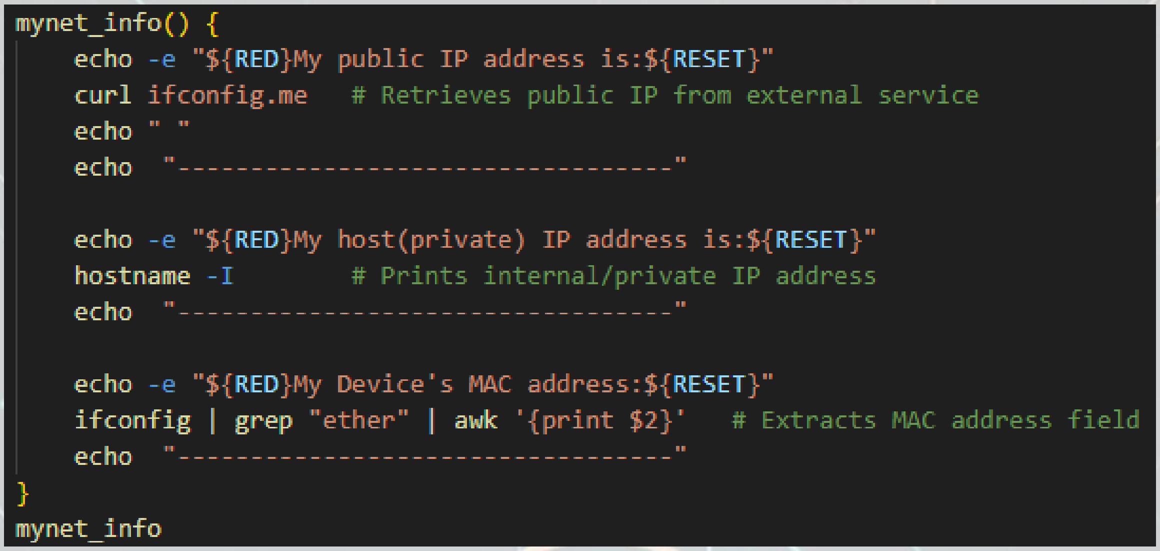
Basic ANSI Color Codes:

- **Foreground Colors (text color):**
    - Black: \e[30m
    - Red: \e[31m
    - Green: \e[32m
    - Yellow: \e[33m
    - Blue: \e[34m
    - Magenta: \e[35m
    - Cyan: \e[36m
    - Light Gray: \e[37m
- **Background Colors:**
    - Black: \e[40m
    - Red: \e[41m
    - Green: \e[42m
    - Yellow: \e[43m
    - Blue: \e[44m
    - Magenta: \e[45m
    - Cyan: \e[46m
    - Light Gray: \e[47m

Other Formatting Options:

- **Reset all attributes:** \e[0m (important for preventing colors from bleeding into subsequent output)
- **Bold/Bright:** \e[1m
- **Underline:** \e[4m

2. Network Information **-Function 1: mynet_info() –**



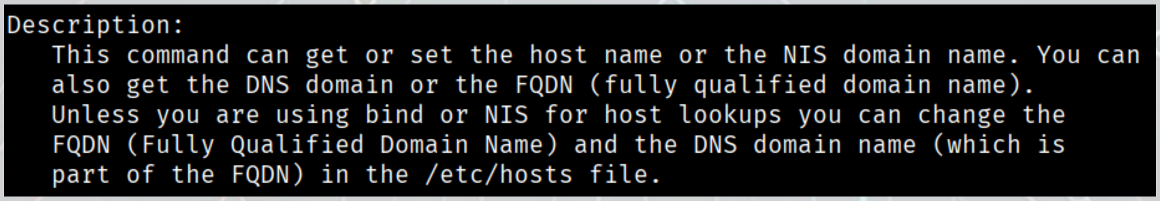
```
mynet_info() {
    echo -e "${RED}My public IP address is:${RESET}"
    curl ifconfig.me   # Retrieves public IP from external service
    echo " "
    echo  "---------------------------------"

    echo -e "${RED}My host(private) IP address is:${RESET}"
    hostname -I        # Prints internal/private IP address
    echo  "---------------------------------"

    echo -e "${RED}My Device's MAC address:${RESET}"
    ifconfig | grep "ether" | awk '{print $2}'   # Extracts MAC address field
    echo  "---------------------------------"
}
mynet_info
```

Purpose: Displays the system's **network identity**

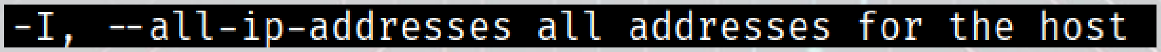
curl ifconfig.me - Queries an external service to display the system's public IP address.
ifconfig.me – Is a webservice that displays information about your connection , including your IP address , hostname ,etc …
Additional alternatives to display network info using curl ifconfig.me

- -6 or --ipv6: Force curl to use IPv6 addresses only.
  Ex: curl -6 ifconfig.me
- **curl ifconfig.me/host**
  This will return your public IPv4 address. Public Hostname.

Hostname -I - Shows the system's **local/private IP address** assigned by the network.

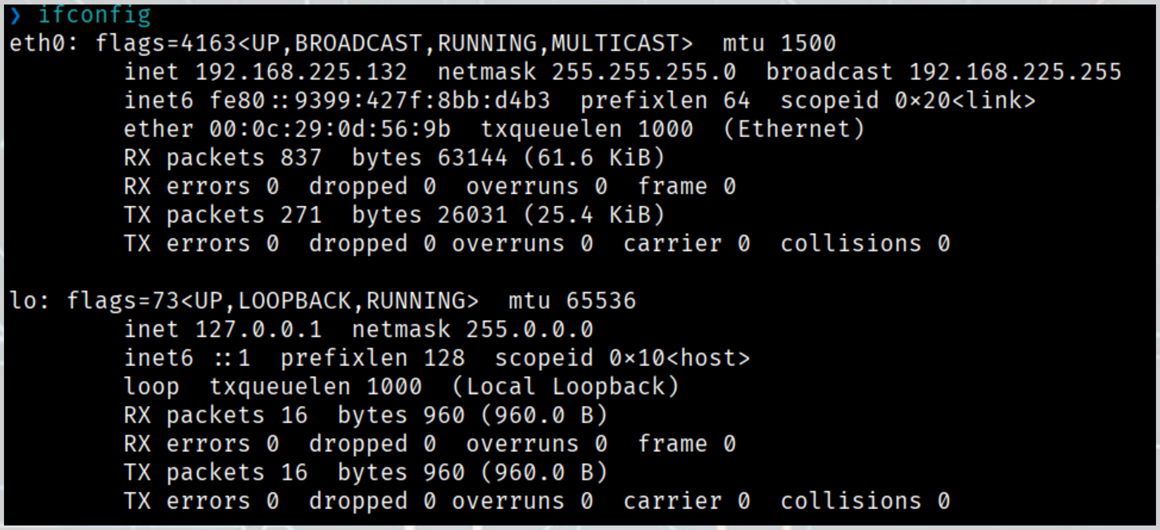


```
Description:
   This command can get or set the host name or the NIS domain name. You can
   also get the DNS domain or the FQDN (fully qualified domain name).
   Unless you are using bind or NIS for host lookups you can change the
   FQDN (Fully Qualified Domain Name) and the DNS domain name (which is
   part of the FQDN) in the /etc/hosts file.
```



```
-I, --all-ip-addresses all addresses for the host
```

Ifconfig – ifconfig, short for **interface configuration**, is a command-line utility in Unix-like operating systems (like Linux) used to display and configure network interface parameters
Can reveal a lot of network information but we used

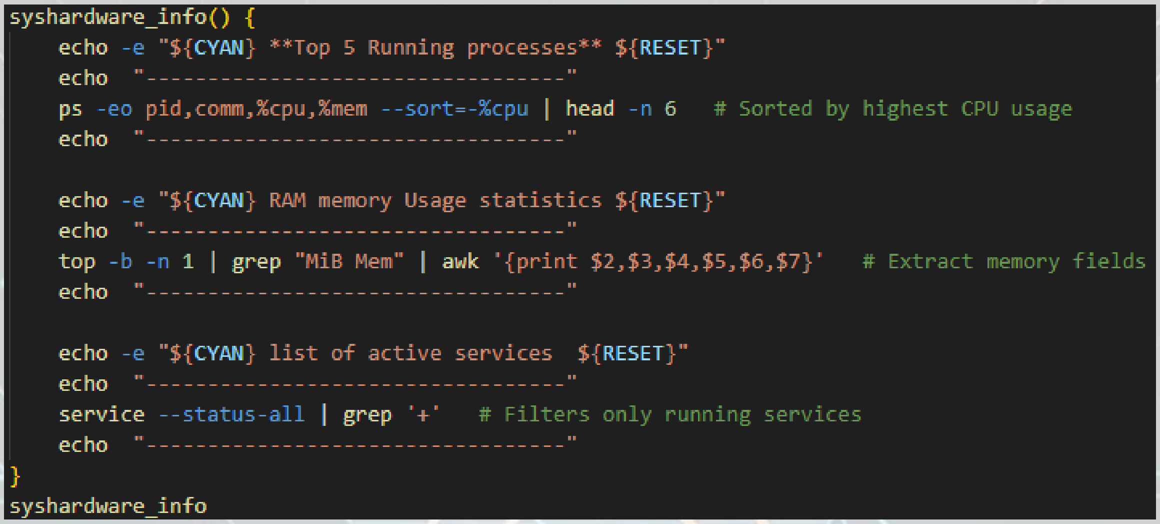


```
❯ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.225.132  netmask 255.255.255.0  broadcast 192.168.225.255
        inet6 fe80::9399:427f:8bb:d4b3  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:0d:56:9b  txqueuelen 1000  (Ethernet)
        RX packets 837  bytes 63144 (61.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 271  bytes 26031 (25.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 16  bytes 960 (960.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 16  bytes 960 (960.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

grep "ether" and then awk '{print $2}' to give us specifically the MAC address

3. System Hardware & Performance **-Function 2: syshardware_info()**

–



```
syshardware_info() {
    echo -e "${CYAN} **Top 5 Running processes** ${RESET}"
    echo  "---------------------------------"
    ps -eo pid,comm,%cpu,%mem --sort=-%cpu | head -n 6   # Sorted by highest CPU usage
    echo  "---------------------------------"

    echo -e "${CYAN} RAM memory Usage statistics ${RESET}"
    echo  "---------------------------------"
    top -b -n 1 | grep "MiB Mem" | awk '{print $2,$3,$4,$5,$6,$7}'   # Extract memory fields
    echo  "---------------------------------"

    echo -e "${CYAN} list of active services  ${RESET}"
    echo  "---------------------------------"
    service --status-all | grep '+'   # Filters only running services
    echo  "---------------------------------"
}
syshardware_info
```

ps -eo pid,comm,%cpu,%mem --sort=-%cpu | head -n 6

ps - Program name. Short for **process status**. It prints a snapshot of processes running on the system.

ps -eo Lists **all running processes**, showing:

- their **PID** (Process ID)
- **command name**
- **CPU usage**
- **memory usage**

Then sorts them by CPU usage (from highest to lowest), showing only the **top 5 processes** (plus the header line)

--sort=-%cpu

- **A long-form option (--sort=) that tells ps how to order the output.**
- **-%cpu is the key:The - (minus)** means descending order (biggest first).%cpu is the field to sort by (CPU usage).
- **So --sort=-%cpu** → sort processes by **%cpu** from largest CPU use to smallest.

3.1 Memory

top -b -n 1 | grep "MiB Mem" | awk '{print $2,$3,$4,$5,$6,$7}'

Displays **memory usage statistics** (RAM) from the system — showing how much memory is total, used, free, and cached.

The **top** program provides a dynamic real-time view of a running system. It can display system summary information as well as a list of processes or threads currently being managed by the Linux kernel. The types of system summary information shown and the types, order and size of information displayed for processes are all user configurable and that configuration can be made persistent across restarts.

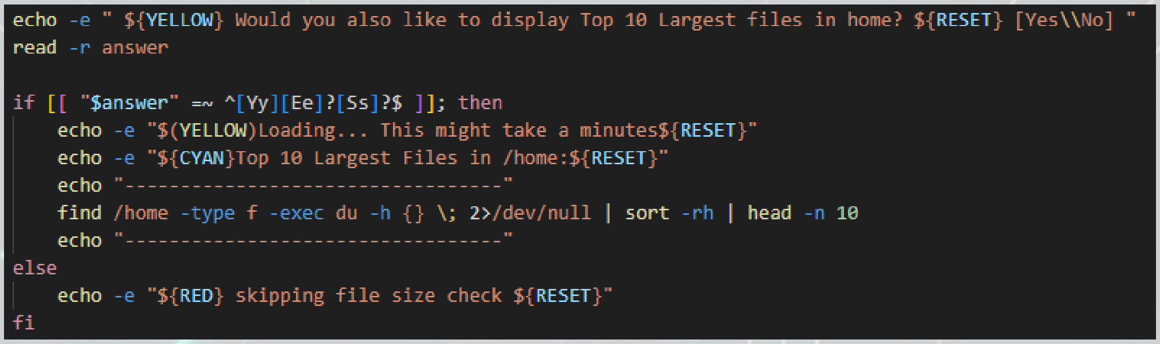
-b By default, when you just type top, it runs in interactive mode (updates every few seconds until you quit).
We don't want interactive output inside a script, so… -b is batch mode

-n 1 The -n flag means "number of iterations".
-n 1 tells top to only update once and then exit.Without it, top might keep refreshing in batch mode forever.
grep "MiB Mem"
we need memory statistics so filter it, will match our task for what we looking for

service --status-all | grep '+'

Lists all system services, showing which ones are active (running).It quickly identifies which background services or daemons are currently active — useful for system security, troubleshooting, and --monitoring.
For example, if a service like SSH is running when it shouldn't be, this helps detect that.

4. Interactive Section – **Large Files in /home**



```
echo -e " ${YELLOW} Would you also like to display Top 10 Largest files in home? ${RESET} [Yes\\No] "
read -r answer

if [[ "$answer" =~ ^[Yy][Ee]?[Ss]?$ ]]; then
    echo -e "$(YELLOW)Loading... This might take a minutes${RESET}"
    echo -e "${CYAN}Top 10 Largest Files in /home:${RESET}"
    echo "----------------------------------"
    find /home -type f -exec du -h {} \; 2>/dev/null | sort -rh | head -n 10
    echo "----------------------------------"
else
    echo -e "${RED} skipping file size check ${RESET}"
fi
```

Gives the user an optional interactive choice to scan for large files.

This script asks yes/no question

Checks if the answer looks like “yes” (in almost any form — yes, y, Y, Yes, YeS, etc.).

If yes → finds and prints the **Top 10 largest files** in the /home directory.

read -r answer = reads the input from the user and stores it as a variable named answer
if Starts a conditional statement
[[...]] lets you use regex matching with =~, and you don’t have to quote some things the same way as with [
"$answer" A reference to the shell variable named answer.
from the read -r answer
which is basically the user’s future to be input

=~ The ~ is actually part of the operator =~, which performs a regular expression match of the string to its left to the extended regular expression on its right.

^ → **start-anchor**: match must begin here (start of string).
[Yy] → **character class**, match either uppercase Y or lowercase y
[Ee]? → character class E or e, followed by ? which means **optional**
[Ss]? → S or s, optional as well.
$ → **end-anchor**: match must end here

find /home -type f -exec du -h {} \; 2>/dev/null | sort -rh | head -n 10

Searches inside /home for all files (not directories)
/home – stating the location to search for
-type f – f stands for file , so we are stating to look for file types

-exec du -h {} \; means For each file found, execute the command du -h on it."
du = also short cut for disk usage

2>/dev/null

Redirects error messages so they wont be displayed

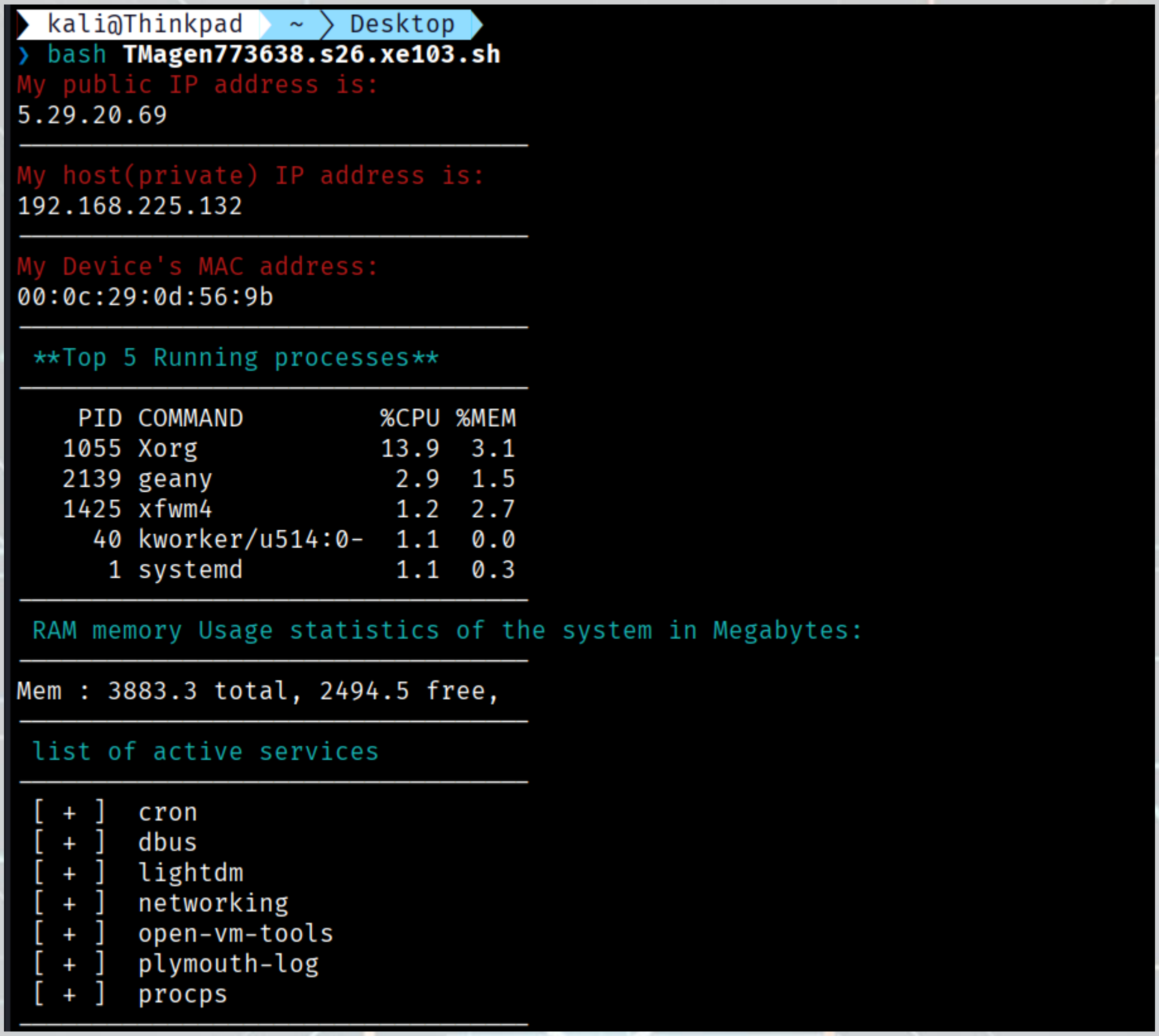
| sort -rh

Takes the list of file sizes from du and sorts them.

And finally **| head -n 10**

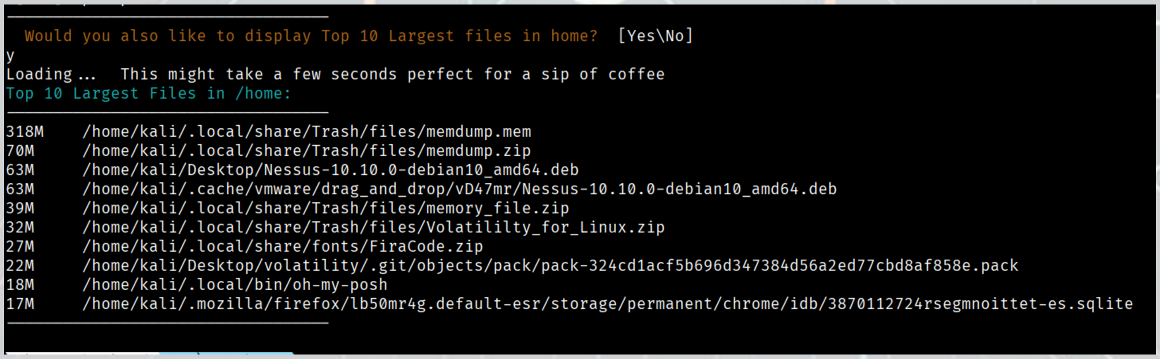
Shows only the **top 10** largest results.

Script output when executed:



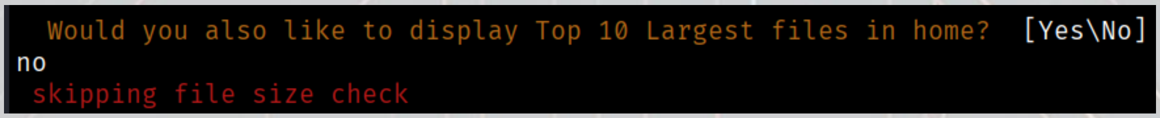
```
 kali@Thinkpad  ~  Desktop
❯ bash TMagen773638.s26.xe103.sh
My public IP address is:
5.29.20.69
__________________________________
My host(private) IP address is:
192.168.225.132
__________________________________
My Device's MAC address:
00:0c:29:0d:56:9b
__________________________________
 **Top 5 Running processes**
__________________________________
    PID COMMAND         %CPU %MEM
   1055 Xorg            13.9  3.1
   2139 geany            2.9  1.5
   1425 xfwm4            1.2  2.7
     40 kworker/u514:0-  1.1  0.0
      1 systemd          1.1  0.3
__________________________________
 RAM memory Usage statistics of the system in Megabytes:
__________________________________
Mem : 3883.3 total, 2494.5 free,
__________________________________
 list of active services
__________________________________
 [ + ]  cron
 [ + ]  dbus
 [ + ]  lightdm
 [ + ]  networking
 [ + ]  open-vm-tools
 [ + ]  plymouth-log
 [ + ]  procps
__________________________________
```

Second section:



```
__________________________________
  Would you also like to display Top 10 Largest files in home?  [Yes\No]
y
Loading ...  This might take a few seconds perfect for a sip of coffee
Top 10 Largest Files in /home:
__________________________________
318M    /home/kali/.local/share/Trash/files/memdump.mem
70M     /home/kali/.local/share/Trash/files/memdump.zip
63M     /home/kali/Desktop/Nessus-10.10.0-debian10_amd64.deb
63M     /home/kali/.cache/vmware/drag_and_drop/vD47mr/Nessus-10.10.0-debian10_amd64.deb
39M     /home/kali/.local/share/Trash/files/memory_file.zip
32M     /home/kali/.local/share/Trash/files/Volatililty_for_Linux.zip
27M     /home/kali/.local/share/fonts/FiraCode.zip
22M     /home/kali/Desktop/volatility/.git/objects/pack/pack-324cd1acf5b696d347384d56a2ed77cbd8af858e.pack
18M     /home/kali/.local/bin/oh-my-posh
17M     /home/kali/.mozilla/firefox/lb50mr4g.default-esr/storage/permanent/chrome/idb/3870112724rsegmnoittet-es.sqlite
__________________________________
```

Or if you chose "no"



```
  Would you also like to display Top 10 Largest files in home?  [Yes\No]
no
 skipping file size check
```

How this project contributed me and challenges faced

During the development of this script, I found myself repeatedly referring to the manuals of different commands to ensure I was using the correct flags, syntax, and combinations—especially when working with text manipulation. Each command and flag behaves differently and can change the output completely, so using them properly saves a lot of troubleshooting and makes later processing much easier.

This process strengthened both my understanding and my memorization of Bash and Linux. I realized how many flags and command variations exist, and how easily using the wrong one can lead to incorrect output or errors. Because of that, I learned that the only real way to build reliable skills in Linux scripting and Bash is through consistent hands-on practice. The more commands I tested, broke, and fixed, the faster I became at recognizing patterns and selecting the right tools for each task.

Overall, the project significantly improved my confidence in Linux, deepened my technical knowledge, and helped me develop a more systematic approach to problem-solving.