Linux Fundamentals | Project: Info Extractor

Kenneth Wong

CFC190324

Trainer: Samson

Table of Contents

Introduction

It is important to have access to crucial information to ensure the health and performance of our devices. Automating these tasks through a Bash script streamlines the process, saves time, reduces the chance of human error, and allows for easy access to information whenever needed.

This report looks into the process of developing of a Bash script for the following functions; aimed at automating the retrieval of useful system information.

1. Identify the system's public IP.
2. Identify the private IP address assigned to the system's network interface.
3. Display the MAC address (masking sensitive portions for security).
4. Display the percentage of CPU usage for the top 5 processes.
5. Display memory usage statistics: total and available memory.
6. List active system services with their status.
7. Locate the Top 10 Largest Files in /home.

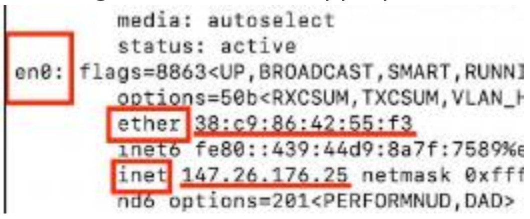
Methodologies

The approaches used for the first three functions were relatively straightforward, as they align closely with methodologies we have previously encountered or adapted in our experience .

1. Identify the system's public IP.
2. Identify the private IP address assigned to the system's network interface.
3. Display the MAC address (masking sensitive portions for security).

Understanding the curl -s ifconfig.co command required researching the curl utility to clarify its function. This involved investigating how curl retrieves data from a server, as its specific application in this context was not initially clear.

The commands chosen for these tasks were generally familiar, with the exception of the section involving the selection of appropriate commands to mask part of the MAC address.



```
        media: autoselect
        status: active
en0: flags=8863<UP,BROADCAST,SMART,RUNNI
        options=50b<RXCSUM,TXCSUM,VLAN_H
        ether 38:c9:86:42:55:f3
        inet6 fe80::439:44d9:8a7f:7589%e
        inet 147.26.176.25 netmask 0xfff
        nd6 options=201<PERFORMNUD,DAD>
```

After extensive online research, a suitable solution could not be found until utilizing ChatGPT. This process involved numerous attempts to refine the prompt and test the generated output to achieve the desired results.

Some of the code snippets encountered were not readily available on websites like Stack Overflow. We opted not to use these codes directly, as we preferred to fully understand their functionality and implications before implementation, rather than adopting them without a comprehensive grasp of how they operate.

Despite these efforts, the initial output was unsatisfactory and required several adjustments before the code functioned as intended.

It became apparent that the tool did not fully grasp the requirements. In one instance, it produced an output of 'XX:XX:XX:XX:XX' when executed. In another case, it inserted 'toupper' into the script, which inadvertently converted the entire text to uppercase. Removing this addition allowed the script to function as intended.

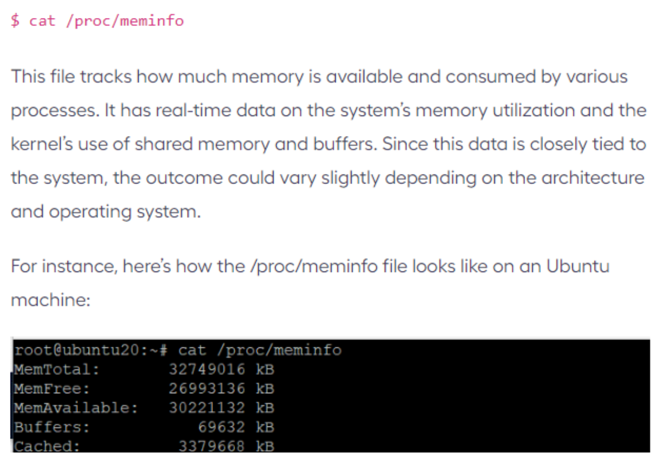
4. Display the percentage of CPU usage for the top 5 processes.

5. Display memory usage statistics: total and available memory.

For the above functions, basic commands were initially employed to achieve the desired results. Exploration and experimentation with these commands were undertaken before seeking information from websites to accurately retrieve CPU usage and available memory through Linux commands.

The '*/proc/meminfo*' is a virtual file which provides information about memory usage and statistics, and the '*top*' command displays real-time information about running processes and system performance. It was relatively straightforward to do up the script for the CPU usage once we understood how '*top*' worked. It was a matter of presenting the information which is clear and readable for the user.

As for '/proc/meminfo', we had to search the web for what had to be performed to attain information on the memory use. With that, it only provided us information on the memory which is free and available, which meant we had to do some arithmetic on how to derive on the memory used which led to more OSINT on the web.

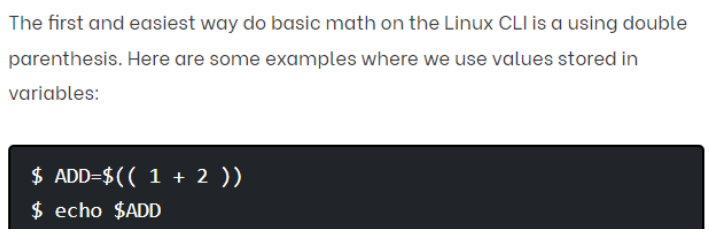


```
$ cat /proc/meminfo
```

This file tracks how much memory is available and consumed by various processes. It has real-time data on the system's memory utilization and the kernel's use of shared memory and buffers. Since this data is closely tied to the system, the outcome could vary slightly depending on the architecture and operating system.

For instance, here's how the /proc/meminfo file looks like on an Ubuntu machine:

```
root@ubuntu20:~# cat /proc/meminfo
MemTotal:        32749016 kB
MemFree:         26993136 kB
MemAvailable:    30221132 kB
Buffers:            69632 kB
Cached:           3379668 kB
```



The first and easiest way do basic math on the Linux CLI is a using double parenthesis. Here are some examples where we use values stored in variables:

```
$ ADD=$(( 1 + 2 ))
$ echo $ADD
```

6. List active system services with their status.

For the above tasks, it involved applying learned methodologies and utilizing systemctl to view the list of active system services along with their status. To achieve this, some OSINT was conducted to gather the necessary information on using systemctl effectively.

7. Locate the Top 10 Largest Files in /home.

Lastly, by using the find approach, we was able to search for the files which I needed which were located in *'/home'*. '*type -f*' allowed us to search specifically for files and '*-printf '%s %p \n*' allowed the files to be output in a specific format. In this case, '*%s*' is used to represent the size of a file in bytes, '*%p*' represents the path of the file, and '*\n*' nearly organises the information allowing it to be printed on a new line.

We understand that '*du*' works for this, but ultimately decided to go with using the '*find*' command as that is what we are more familiar with.

Discussion

1. Identify the system's public IP.

```
curl -s ifconfig.co
```

This command retrieves the public IP address of the system using the curl utility.

'curl' is a command for transferring data from or to a server.

'-s' instructs curl to operate in silent mode, suppressing progress or error messages.

'ifconfig.co' is the URL from which the command fetches the public IP address.

2. Identify the private IP address assigned to the system's network interface.

```
ifconfig | grep broadcast | awk '{print$2}'
```

This command retrieves the internal IP address of the machine.

‘grep broadcast’ filters the output of ifconfig to only include lines containing the word "broadcast".

‘awk ‘{print$2}’’: Extracts the second field which contains the IP address from the filtered output.

3. Display the MAC address (masking sensitive portions for security).

```
ifconfig | grep ether | awk '{split($2, a, ":"); print("XX:XX:XX:" a[4] ":" a[5] ":" a[6])}'
```

This command retrieves the MAC address of the machine and censors the first part.

'grep ether' filters the output of ‘ifconfig’ to only include lines containing the word "ether" which is on the same line as MAC address.

'split($2, a, ":")' splits the second field of the MAC address by ":" and stores the parts in array "a".

'print("XX:XX:XX:" a[4] ":" a[5] ":" a[6])' prints the censored MAC address, replacing the first three parts with "XX:XX:XX".

4. Display the percentage of CPU usage for the top 5 processes.

```
top | head -n12 | tail -n5 | awk '{print$(NF-4),$(NF-1)}'
```

This command retrieves the top 5 processes' CPU usage and % from the output of the 'top' command.

'top': Command to display dynamic real-time information about running processes.

5. Display memory usage statistics: total and available memory.

```
mem_avail=$(cat /proc/meminfo | grep -i memavailable | awk '{print$2}')

mem_free=$(cat /proc/meminfo | grep -i memfree | awk '{print$2}')

used_memory=$((mem_avail - mem_free))
```

These commands retrieve the value of available and free memory from the /proc/meminfo file.

Using arithmetic, I was able to determine the used memory.

6. List active system services with their status.

```
sudo systemctl list-units --type=service --state=active
```

Displays a list of all active service units on the system, providing information about which services are currently running.

7. Locate the Top 10 Largest Files in /home.

```
find /home -type f -printf '%s %p \n' | sort -nr | head -10
```

This command finds files in the /home directory and prints their sizes and paths.

'find /home' searches for files within the /home directory.

'-type f' specifies that only regular files should be included in the search.

'-printf '%s %p\n'' specifies the format for the output.

'%s' prints the size of each file.

'%p' prints the file path.

'\n' adds a newline character after each entry.

Conclusion

Creating Bash scripts represents a significant advancement in streamlining administrative tasks by consolidating the retrieval of key system information into a single process. This approach enhances the management of computing environments by providing greater control and visibility.

The development of this script proved to be an invaluable learning experience, underscoring the importance of comprehensive documentation for each line of code. This practice not only facilitates future understanding and modifications but also enhances clarity for others reviewing the script.

Additionally, the process reinforced the need for rigorous testing to ensure that each command functions as intended and delivers accurate results.

Overall, the automation of system information retrieval through Bash scripting contributes to increased efficiency and effectiveness in managing computing resources.

Recommendation

The script could be further improved by integrating it with specific tools that trigger notifications based on predefined configurations and conditions, such as high resource usage or the activation of unknown services.

These alerts should be informative and actionable, including timestamps and relevant details to facilitate efficient troubleshooting by administrators. Additionally, incorporating platform-agnostic compatibility would enhance the script's versatility, allowing it to function across various operating systems.

Further recommendations include implementing log management to record actions and outputs for historical analysis, introducing user-configurable parameters to tailor the script's behaviour, enhancing error handling to manage unexpected conditions gracefully, optimizing performance to ensure efficiency under heavy workloads, and adding authentication features to secure sensitive operations. These enhancements will create a more robust and flexible tool, better suited for diverse administrative tasks and environments.

References

1. MAC Address: Texas State University. "IP Address Reservation for Linux." https://itac.txst.edu/support/ipaddress-reservation/linux.html.

2. Memory Usage: RedSwitches. "How to Check Linux Memory Usage." https://www.redswitches.com/blog/check-linux-memory-usage/.

3. Arithmetic in Terminal: Tecmint. "Arithmetic Operations in Linux Terminal." https://www.tecmint.com/arithmetic-in-linux-terminal/.

4. Censor Text with Regex: Stack Exchange. "Censor Text with Regex." https://unix.stackexchange.com/questions/223584/censor-text-with-regex.

5. Finding Largest Files: Tutorialspoint. "Find the Largest (Top 10) Files and Directories on a Linux." https://www.tutorialspoint.com/find-the-largest-top-10-files-and-directories-on-a-linux.

6. Finding Largest Files: Cyberciti. "Linux Find Largest File in Directory Recursively Using Find/Du." https://www.cyberciti.biz/faq/linux-find-largest-file-in-directory-recursively-using-find-du/.