



**Module Code & Module Title**  
**Level 5 – Network Operating System**

**Assessment Type**  
**Logbook 8**  
**Semester 1**  
**2023/2024 Autumn**

**Student Name:** Dipen Khatri  
**London Met ID:** 23056968  
**College ID:** NP04CP4A230133  
**Assignment Due Date:** 28 December 2024  
**Assignment Submission Date:** 28 December 2024  
**Submitted To:** Mr. Prasant Adhikari

*I confirm that I understand my coursework needs to be submitted online via GitHub before the deadline for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.*

## Table of Contents

Introduction .....	1
VirtualBox .....	2
Kali Linux .....	3
Terminal and Shell Commands: .....	4
Q.1: .....	5
Q.2 .....	5
Q.3 .....	5
Q.4 .....	6
Q.5 .....	9
Q.6 .....	11
Q.7 .....	12
Q.8 .....	12
Q.9 .....	13
Q.10 .....	14
Q.11 .....	15
Q.12 .....	15
Q.13 .....	16
Conclusion .....	17
References .....	18

## Table of figures

Figure 1: Virtual Box .....	2
Figure 2: Kali Linux .....	3
Figure 3: Terminal .....	4
Figure 4: Creating the Directory Structure .....	5
Figure 5: Change the directory .....	5
Figure 6: Creating testa file.....	6
Figure 7: Creating testb file.....	6
Figure 8: grep ll testa .....	7
Figure 9: grep -v ll testa .....	7
Figure 10: grep -n ll testa .....	7
Figure 11: grep -l ll * .....	8
Figure 12: grep -i ll * .....	8
Figure 13: grep -i LL * .....	8
Figure 14: grep -c ll * .....	9
Figure 15: grep '^k' testa testb .....	9
Figure 16: grep -n '^' testa .....	9
Figure 17: Isal alias .....	10
Figure 18: Removing the alias .....	11
Figure 19: Re-defining the alias .....	12
Figure 20: Defining nwho alias.....	12
Figure 21: nwho command .....	13
Figure 22: History.....	14
Figure 23: re- Executing command.....	15
Figure 24: Re-executing the three commands ago.....	15
Figure 25: fc -s l .....	16

## Introduction

In this log, we focused on practicing several essential UNIX utilities to enhance our ability to manage files, search through text, and efficiently handle command-line tasks. These utilities are fundamental for anyone working in a UNIX-based environment, especially when dealing with file manipulation, text processing, and system administration. To complete these tasks, we used Kali Linux running on VirtualBox, which provided us with a controlled environment for executing commands and exploring the UNIX tools.

During this session, we explored the **grep** command, which we used to search for specific text patterns within files. We applied various options to refine our searches, such as displaying line numbers, performing case-insensitive searches, and excluding certain lines from the results. Additionally, we defined and managed custom aliases, which allowed us to simplify frequently used commands, making our workflow more efficient. We also experimented with command history, learning how to access and re-execute previous commands using the **history** and **fc** utilities.

The goal was to understand how these tools can automate repetitive tasks, improve productivity, and streamline our command-line operations. By the end of the log, we gained hands-on experience with these utilities, enabling us to perform tasks more efficiently in any UNIX-based system. This exercise was foundational for tasks like system administration, scripting, and general command-line proficiency. It also gave us valuable experience using Kali Linux and VirtualBox, which are both powerful tools for learning and practicing UNIX and security-related tasks.

## VirtualBox

**VirtualBox** is an open-source virtualization software that allows users to run multiple operating systems (OSes) on a single physical machine. It acts as a hypervisor, creating virtual machines (VMs) that simulate physical hardware, enabling users to install and run various OSes on top of their host system. It is commonly used for testing, development, and running different operating systems concurrently without needing additional hardware (Corporation, 2024).

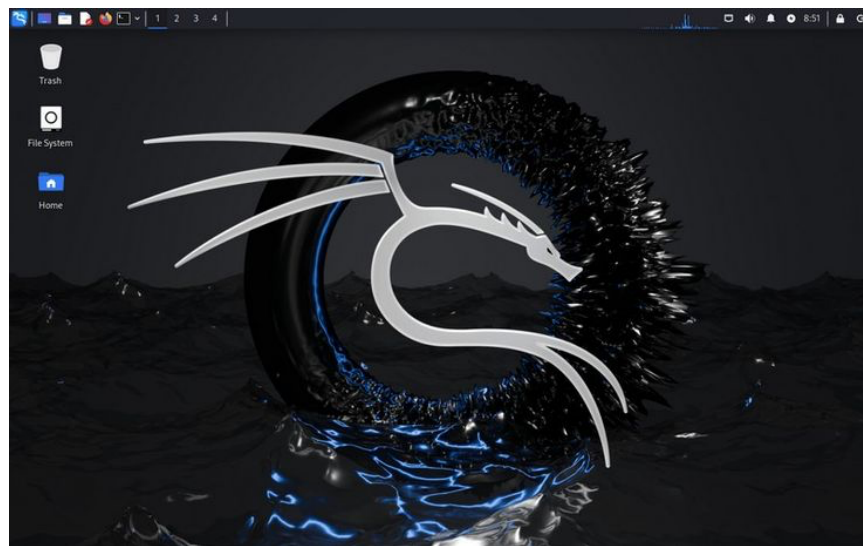


*Figure 1: Virtual Box*

In our log, we used VirtualBox to create a virtual machine (VM) where Kali Linux is installed. This allows us to run Kali Linux in a controlled, isolated environment without modifying our main operating system. By using VirtualBox, we can safely execute various Linux commands and tasks, such as file management and system operations, in Kali Linux. This setup provides a secure platform for learning and experimenting with Linux without any risk to our host system.

## Kali Linux

**Kali Linux** is a Debian-based Linux distribution specifically geared towards penetration testing, ethical hacking, and network security assessments. It comes pre-installed with numerous security and forensics tools, making it popular among cybersecurity professionals and enthusiasts. Kali Linux is developed and maintained by Offensive Security, and it's widely used for tasks like vulnerability scanning, exploitation, and network monitoring (Security, 2024).

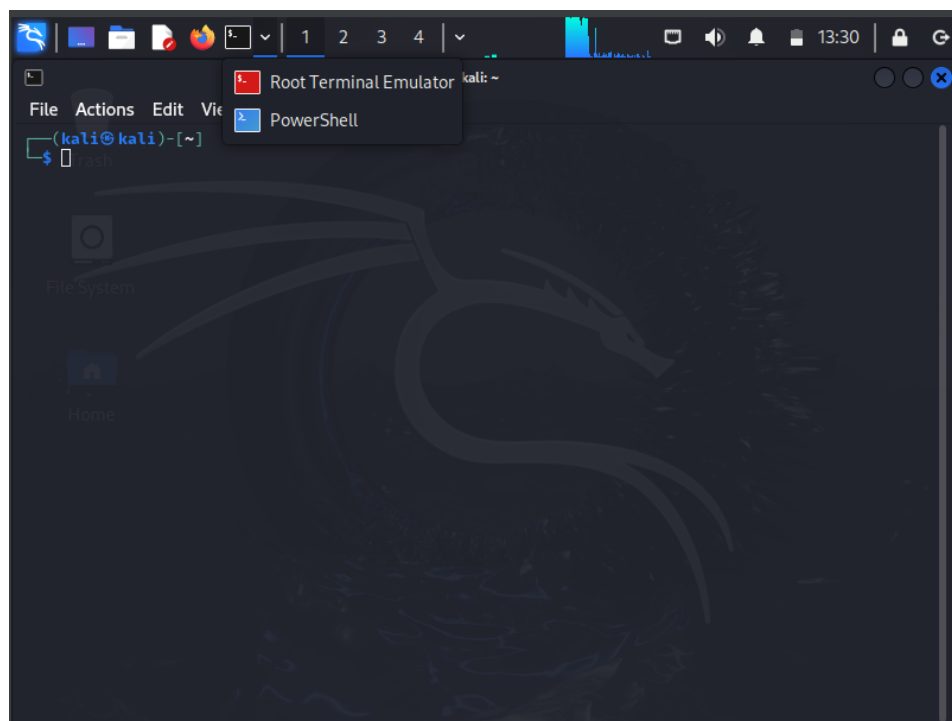


*Figure 2: Kali Linux*

In our log, we use Kali Linux as the operating system to perform various Linux commands and file management tasks. Kali Linux provides us with the tools to interact with the terminal and execute commands such as `whoami`, `ls`, `date`, and `cat`, allowing us to view system information, list files, and manipulate them. Through Kali Linux, we can safely experiment and practice these tasks in a controlled environment. This allows us to gain hands-on experience with system administration while ensuring that any potential errors do not affect the main operating system.

## Terminal and Shell Commands:

Terminal and shell commands in Linux are essential tools for interacting with the system through text-based input. They allow users to quickly perform tasks, from retrieving system information (like `whoami` for the username) to managing files and directories (like `ls` for listing files, `cat` for viewing or creating file content). These commands are highly efficient for navigating the system, creating or modifying files, and accessing critical information about the system's state, making them indispensable for users working with Linux environments.



*Figure 3: Terminal*

**Q.1:** Create the directory structure as presented in the figure.

```
(kali㉿kali)-[~]  
$ ls W8  
8cat-grep  
  
(kali㉿kali)-[~]  
$ mkdir -p W8/8cat-grep
```

*Figure 4: Creating the Directory Structure*

The command **mkdir -p W8/8cat-grep** creates a directory structure where **8cat-grep** is a subdirectory inside **W8**. The **-p** option ensures that if the **W8** directory doesn't exist, it will be created first, and then **8cat-grep** will be created inside it. This allows the entire structure to be created in one step.

**Q.2** Change to the **8cat-grep** directory by one step using a relative pathname.

```
(kali㉿kali)-[~]  
$ cd W8/8cat-grep  
  
(kali㉿kali)-[~/W8/8cat-grep]  
$
```

*Figure 5: Change the directory*

**CD** commands help to change the directory from home to **8cat-grep**.

**Q.3** Using the cat utility, create two files.

File testa	File testb
Kkkll	KKKKK
lllmm	LLLLL
oo—oo	MMMMM



mmmdd       DDDDD

dddkk

```
(kali㉿kali)-[~/W8/8cat-grep]
$ cat > testa << EOF
heredoc> Kkkll
heredoc> lllmm
heredoc> oo-oo
heredoc> mmmdd
heredoc> dddkk
heredoc> EOF
```

Figure 6: Creating testa file

```
(kali㉿kali)-[~/W8/8cat-grep]
$ cat > testb << EOF
heredoc> KKKKK
heredoc> LLLLL
heredoc> MMMMM
heredoc> DDDDD
heredoc> EOF
```

Figure 7: Creating testb file

Using **EOF** with the **cat** utility allows us to create files and input multi-line content efficiently. The command **cat > filename << EOF** starts a "here document," where everything typed between the **<< EOF** and the ending **EOF** is redirected into the specified file. This method is simple and effective for quickly creating files with multiple lines of text, making it useful for scripting and automation.

**Q.4** Give the following commands and explain the results for yourself.

- **grep ll testa**
- **grep -v ll testa**
- **grep -n ll testa**
- **grep -l ll \***

- `grep -i ll *`
- `grep -i LL *`
- `grep -c ll *`
- `grep '^K' testa testb`
- `grep -n '^' testa`

```
(kali㉿kali)-[~/W8/8cat-grep]
$ grep ll testa
Kkkll
llmm
```

Figure 8: `grep ll testa`

This searches for the string **ll** in the file **testa**. It will display the lines in **testa** that contain **ll**.

```
(kali㉿kali)-[~/W8/8cat-grep]
$ grep -v ll testa
00-00
mmdd
dddkk
```

Figure 9: `grep -v ll testa`

This searches for lines in **testa** that do *not* contain the string **ll**. The **-v** option inverts the match.

```
(kali㉿kali)-[~/W8/8cat-grep]
$ grep -n ll testa
1:Kkkll
2:llmm
```

Figure 10: `grep -n ll testa`

This searches for the string **ll** in **testa** and show the line numbers where **ll** is found. The **-n** option displays line numbers.

```
(kali㉿kali)-[~/W8/8cat-grep]
$ grep -l ll *
testa
```

Figure 11: `grep -l ll *`

This searches for the string `ll` in all files in the current directory and lists only the names of files that contain `ll`. The `-l` option shows filenames.

```
(kali㉿kali)-[~/W8/8cat-grep]
$ grep -i ll *
testa:Kkkll
testa:lllmm
testb:LLLLL
```

Figure 12: `grep -i ll *`

This searches for the string `ll` case-insensitively in all files in the current directory. The `-i` option makes the search case-insensitive.

```
(kali㉿kali)-[~/W8/8cat-grep]
$ grep -i LL *
testa:Kkkll
testa:lllmm
testb:LLLLL
```

Figure 13: `grep -i LL *`

This searches for the string `LL` case-insensitively in all files. The `-i` option allows it to match `ll`, `LL`, `lL`, and `Ll`.

```
(kali㉿kali)-[~/W8/8cat-grep]
$ grep -c ll *
File:0
testa:2
testb:0
```

Figure 14: `grep -c ll *`

This counts the occurrences of **ll** in all files in the current directory. The **-c** option counts the number of matching lines.

```
(kali㉿kali)-[~/W8/8cat-grep]
$ grep '^K' testa testb
testa:Kkll
testb:KKKKK
```

Figure 15: `grep '^k' testa testb`

This searches for lines that start with the letter **K** in both **testa** and **testb**. The **^** symbol indicates the beginning of the line.

```
(kali㉿kali)-[~/W8/8cat-grep]
$ grep -n '^' testa
1:Kkkl
2:llmm
3:oo-oo
4:mmdd
5:ddkk
```

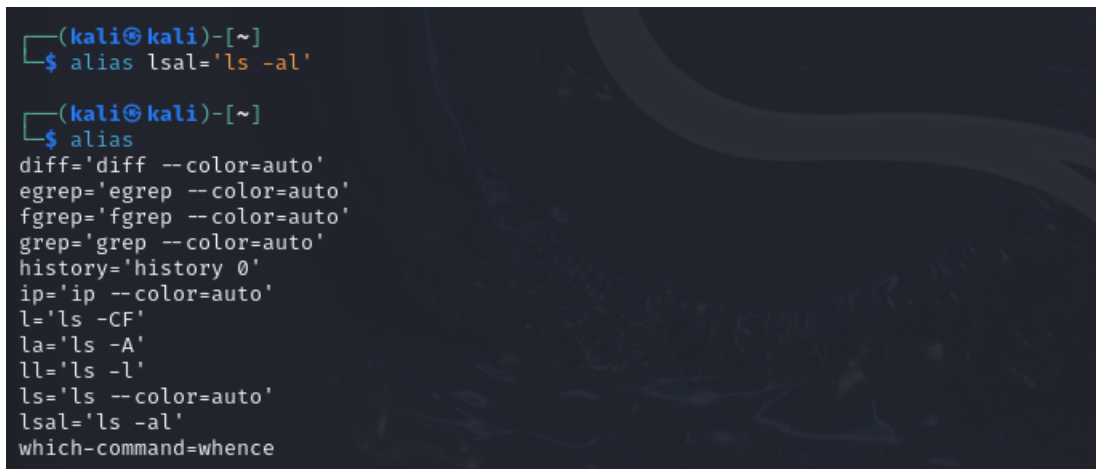
Figure 16: `grep -n '^' testa`

This shows all lines in **testa** with their line numbers, as the **^** symbol matches the beginning of every line. The **-n** option shows line numbers.

### Q.5 Define the **lsal** alias for **ls -al** command

Show that your system stores it giving the **alias** command (without arguments).

Use it in your home directory.

A terminal window with a dark background and light blue/green text. The prompt is (kali@kali)-[~]. The first command is \$ alias lsal='ls -al'. The second command is \$ alias, which lists several aliases: diff='diff --color=auto', egrep='egrep --color=auto', fgrep='fgrep --color=auto', grep='grep --color=auto', history='history 0', ip='ip --color=auto', l='ls -CF', la='ls -A', ll='ls -l', ls='ls --color=auto', lsal='ls -al', and which-command=whence.

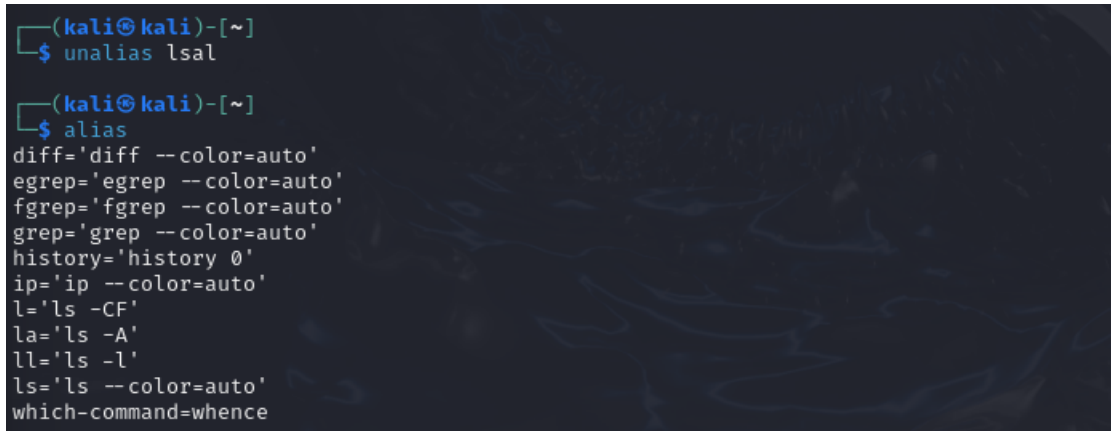
```
(kali@kali)-[~]  
$ alias lsal='ls -al'  
  
(kali@kali)-[~]  
$ alias  
diff='diff --color=auto'  
egrep='egrep --color=auto'  
fgrep='fgrep --color=auto'  
grep='grep --color=auto'  
history='history 0'  
ip='ip --color=auto'  
l='ls -CF'  
la='ls -A'  
ll='ls -l'  
ls='ls --color=auto'  
lsal='ls -al'  
which-command=whence
```

Figure 17: *lsal alias*

An **alias** is a shortcut for a command or series of commands in a UNIX-like system. It allows us to create custom command names or modify existing ones for convenience. For example, **alias lsal='ls -al'** creates an alias called **lsal** that runs the **ls -al** command. When the **alias** is used, the system replaces it with the associated command, making it easier and faster to execute repetitive tasks. **Aliases** are stored temporarily for the current session, and to make them permanent, they can be added to configuration files like **.bashrc**.

**Q.6** Remove the alias.

Show that your system does not store it.

A terminal window with a dark background and light blue text. The prompt is '(kali㉿kali)-[~]'. The first command entered is '\$ unalias lsal'. The second command entered is '\$ alias', which lists several aliases: diff, egrep, fgrep, grep, history, ip, l, la, ll, ls, and which-command. The 'lsal' alias is not present in the list.

```
(kali㉿kali)-[~]
$ unalias lsal

(kali㉿kali)-[~]
$ alias
diff='diff --color=auto'
egrep='egrep --color=auto'
fgrep='fgrep --color=auto'
grep='grep --color=auto'
history='history 0'
ip='ip --color=auto'
l='ls -CF'
la='ls -A'
ll='ls -l'
ls='ls --color=auto'
which-command=whence
```

*Figure 18: Removing the alias*

An **alias** can be removed using the **unalias** command, which deletes the custom shortcut for a command. For example, **unalias lsal** removes the **alias lsal**. Once the **alias** is removed, running the **alias** command will show that it is no longer listed, meaning the system does not store it anymore. **Aliases** are typically temporary and only persist for the current session unless added to configuration files like **.bashrc**. After removing an alias, it will not be available in future sessions unless redefined.

**Q.7** Define this alias again preserving it for the next session.

Show that the system keeps this your alias.

```
(kali㉿kali)-[~]
$ echo "alias lsal='ls -al' " >> ~/.bashrc
(kali㉿kali)-[~]
$ source ~/.bashrc
(kali㉿kali)-[~]
$ alias lsal
alias lsal='ls -al'
(kali㉿kali)-[~]
$ alias lsal='ls -al'
```

*Figure 19: Re-defining the alias*

To make an **alias** persistent across sessions, we define the alias using the alias command, then add it to the **bashrc** file. This ensures that the alias is automatically created every time a new terminal session starts. After adding the alias to **bashrc**, we use `source ~/.bashrc` to apply the changes immediately. This way, the alias will be available in future sessions without needing to redefine it manually each time.

**Q.8** Define the **nwho** alias for the number of system file at UNIX computers.

*alias nwho='getent passwd|wc -l'*

```
(kali㉿kali)-[~]
$ alias nwho='getent passwd|wc -l'
(kali㉿kali)-[~]
$ nwho
57
(kali㉿kali)-[~]
$ uname -a
Linux kali 6.8.11-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.8.11-1kali2 (2024-05-30) x86_64 GNU/Linux
(kali㉿kali)-[~]
$
```

*Figure 20: Defining nwho alias*

The **nwho** alias is defined to count the number of system users (or system files) on a UNIX system. The command **getent passwd** retrieves a list of user accounts from the system's database, and **wc -l** counts the number of lines in the output. When combined in the **alias alias nwho='getent passwd | wc -l'**, it returns the total number of users on the system. This alias can be used to quickly check the number of user accounts without manually running both commands.

**Q.9** Give the command **nwho**.



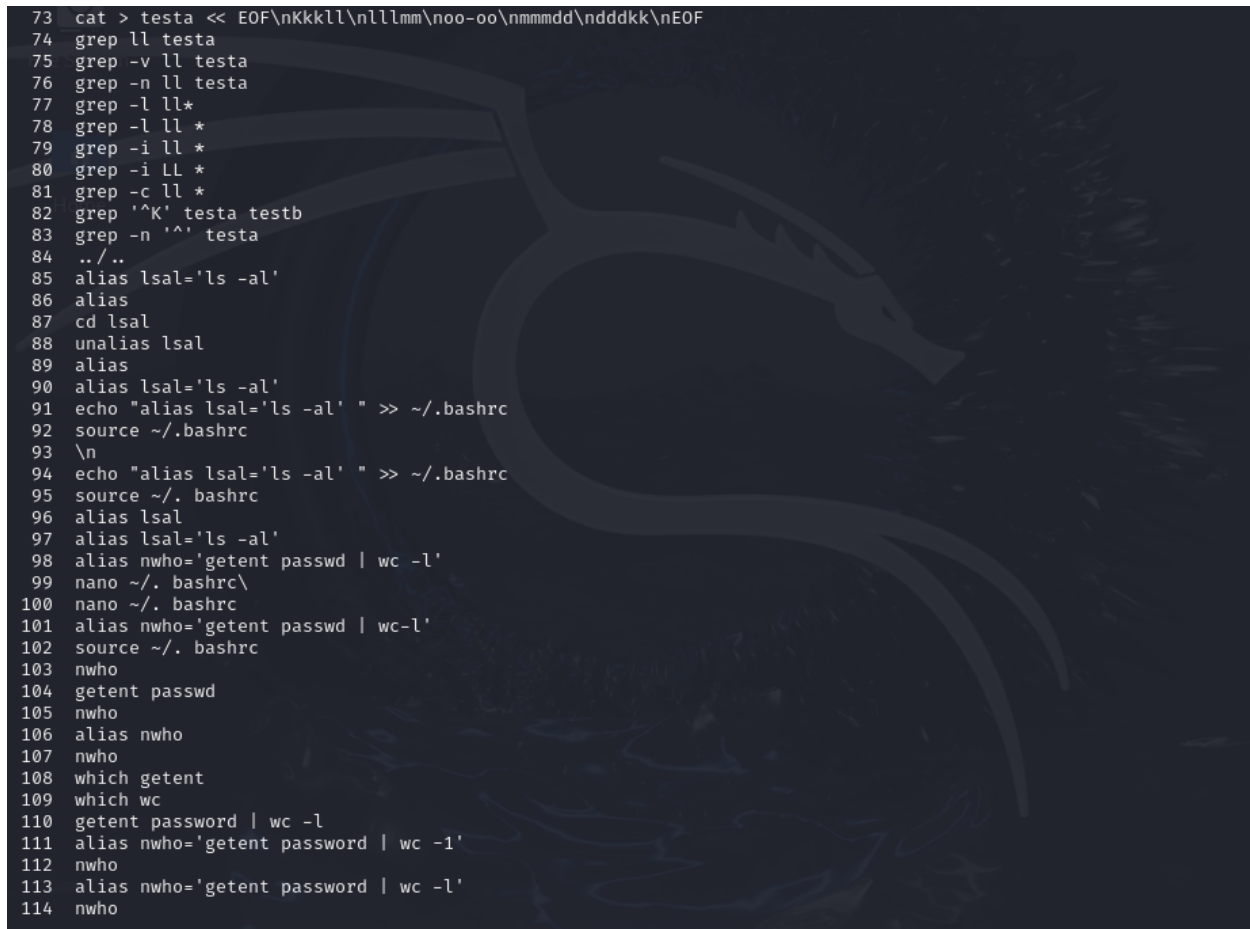
```
(kali@kali)-[~]  
$ nwho  
57
```

*Figure 21: nwho command*

The **nwho** alias counts the number of user accounts on a UNIX-like system.



**Q.10** List your last commands executed giving the **history** command.



```

73 cat > testa << EOF\nKkkll\nlllmm\noo-oo\nmmdd\ndddkk\nEOF
74 grep ll testa
75 grep -v ll testa
76 grep -n ll testa
77 grep -l ll*
78 grep -l ll *
79 grep -i ll *
80 grep -i LL *
81 grep -c ll *
82 grep '^K' testa testb
83 grep -n '^' testa
84 ../..
85 alias lsal='ls -al'
86 alias
87 cd lsal
88 unalias lsal
89 alias
90 alias lsal='ls -al'
91 echo "alias lsal='ls -al' " >> ~/.bashrc
92 source ~/.bashrc
93 \n
94 echo "alias lsal='ls -al' " >> ~/.bashrc
95 source ~/. bashrc
96 alias lsal
97 alias lsal='ls -al'
98 alias nwho='getent passwd | wc -l'
99 nano ~/. bashrc\
100 nano ~/. bashrc
101 alias nwho='getent passwd | wc-l'
102 source ~/. bashrc
103 nwho
104 getent passwd
105 nwho
106 alias nwho
107 nwho
108 which getent
109 which wc
110 getent password | wc -l
111 alias nwho='getent password | wc -l'
112 nwho
113 alias nwho='getent password | wc -l'
114 nwho

```

*Figure 22: History*

The **history** command in UNIX-like systems allows us to view a list of commands we've recently executed. It shows the commands in reverse order, with the most recent ones appearing at the bottom. By default, the list includes all previous commands, but we can limit the output by specifying a number, like `history 10` to show the last 10 commands. This command helps us quickly recall and reuse commands without having to retype them.

**Q.11** Re-execute the *last but one* command using the **redo** (r) command and the number of the event. *fc -r*

A terminal window with a dark background. The prompt is '(kali㉿kali)-[~]'. The user enters '\$ fc 2' and the output is 'cd'.

Figure 23: re- Executing command

The **fc -r** command in UNIX re-executes a previous command based on its event number. The **-r** option runs the command in reverse order, allowing us to quickly repeat past commands from the history.

**Q.12** Re-execute the command given *three commands ago* using the negative integer.

*!-3*

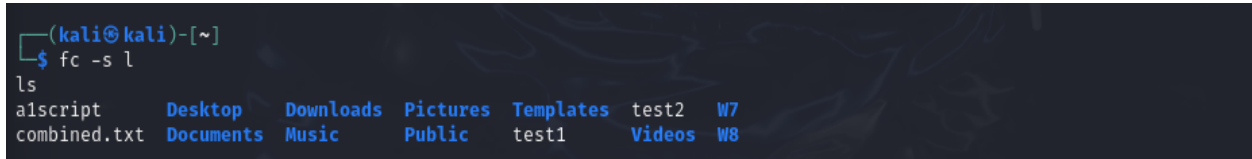
A terminal window with a dark background. It shows a sequence of four commands and their outputs. The first command is '\$ echo "A"' with output 'A'. The second is '\$ echo "B"' with output 'B'. The third is '\$ echo "C"' with output 'C'. The fourth is '\$ !-3' with output 'echo "A"' and 'A'.

Figure 24: Re-executing the three commands ago

The command **!-3** is used to re-execute the command that was run three commands ago. The **!-3** refers to the third command in history before the current one. This allows us to quickly repeat a previous command without having to find or retype it manually.

**Q.13** Re-execute the last command which name begins with 'l'.

*fc -s l*



```
(kali㉿kali)-[~]  
$ fc -s l  
ls  
alscript      Desktop      Downloads    Pictures     Templates   test2    W7  
combined.txt  Documents    Music        Public       test1       Videos   W8
```

*Figure 25: fc -s l*

The command **fc -s -l** is used to re-execute the most recent command that begins with the letter **-l**. The **fc -s** option allows us to directly execute a command from the history without opening it in an editor. By specifying **-l**, we instruct the **fc** command to find the latest command in the history that starts with the letter **l** and immediately execute it. This is a quick way to repeat a specific command without having to search for it manually in history.

## Conclusion

In conclusion, we have explored essential UNIX commands and techniques that enhance our command-line efficiency and system management. Through creating and using aliases like **lsal** and **nwho**, we simplified common tasks and tailored commands for easier access. We also practiced file management using the **cat** command and the **EOF** redirection for efficient content creation. Additionally, by leveraging the **grep** command, we learned to search and manipulate text in files effectively. The history command, along with methods like **fc -r**, **!-3**, and **fc -s**, allowed us to re-execute previous commands quickly, improving workflow. These tools are fundamental in UNIX systems for automating tasks, navigating efficiently, and optimizing our work environment. Understanding these concepts gives us greater control and flexibility in managing tasks, making our interactions with the system faster and more productive.

## References

Corporation, O. (2024, December 13). *VirtualBox*. Retrieved from VirtualBox Official Website:  
<https://www.virtualbox.org/>

Security, O. (2024, December 13). *Kali Linux*. Retrieved from Kali Linux Official Website:  
<https://www.kali.org/>