Linux Fundamentals

XE103

v1.3

THINKCYBER

Table of Contents

Virtualization

Virtualization is a technology that allows you to create multiple simulated environments or dedicated resources from a single, physical hardware system. Essentially, it lets you run multiple operating systems and applications on a single physical machine as if they were running on separate hardware.

Types of Virtualization

1. **Hardware/Platform Virtualization**: This is the most common form, where a hypervisor (like VMware's ESXi, Microsoft's Hyper-V, or Oracle's VirtualBox) runs directly on the physical hardware (Type 1 or bare-metal) or atop an operating system (Type 2 or hosted). The hypervisor creates, runs, and manages multiple virtual machines (VMs) that can have different operating systems.
2. **Desktop Virtualization**: This involves separating the physical machine from the OS to create a virtual desktop that the user can access remotely. Examples include VMware Horizon View and Citrix Virtual Apps and Desktops.
3. **Software Virtualization**: This involves running multiple versions of an application on a single OS, ensuring they don't interfere with each other. Containers, like Docker, are a popular form of software virtualization.
4. **Memory Virtualization**: This pools the physical memory from multiple servers to create a virtualized memory pool shared across multiple machines.
5. **Storage Virtualization**: This pools physical storage from multiple network storage devices so that it appears as a single storage device. It can be done at the block level or the file level.
6. **Network Virtualization**: This divides available bandwidth into independent channels that can be assigned to servers or devices in real-time. It can be categorized as either external (combining many networks into a virtual unit) or internal (breaking network functionality into manageable parts).

Advantages of Virtualization

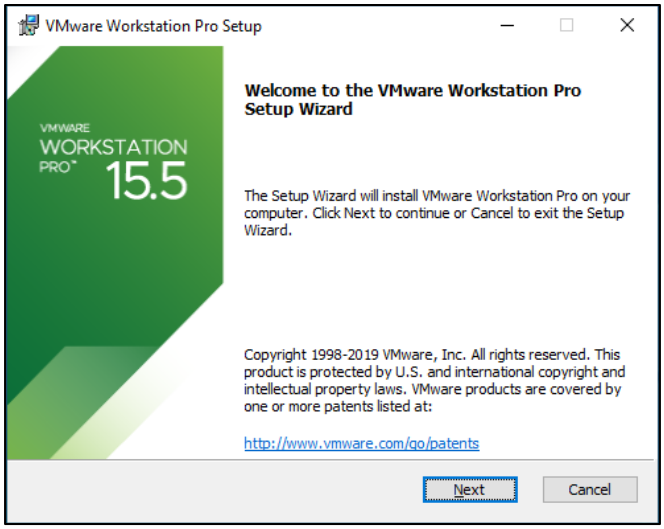
1. **Resource Efficiency**: Multiple VMs can run on a single physical machine, maximizing resource utilization.
2. **Cost Savings**: Reduces the need for physical hardware systems, leading to savings in hardware costs, energy consumption, and maintenance.
3. **Isolation**: VMs are isolated from each other. If one crashes, it doesn't affect others.
4. **Flexibility and Agility**: Quickly deploy, clone, and migrate VMs based on demand.
5. **Snapshot and Cloning**: Capture the state of a VM at a particular point in time, allowing for easy backup and recovery.
6. **Security**: Potential security breaches can be isolated to a particular VM without affecting the host system or other VMs.

Challenges of Virtualization

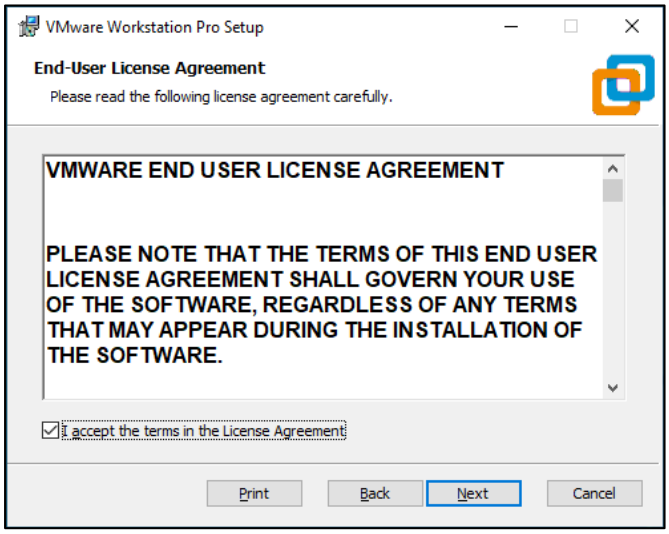
1. **Overhead**: Virtualization introduces a layer of overhead due to the hypervisor.
2. **Complex Management**: Managing virtual environments can become complex, especially at scale.
3. **Security Concerns**: If not properly configured, the hypervisor can be a point of vulnerability.
4. **Licensing**: Software licensing can become complex in virtualized environments.

Installing Workstation on a Windows Host

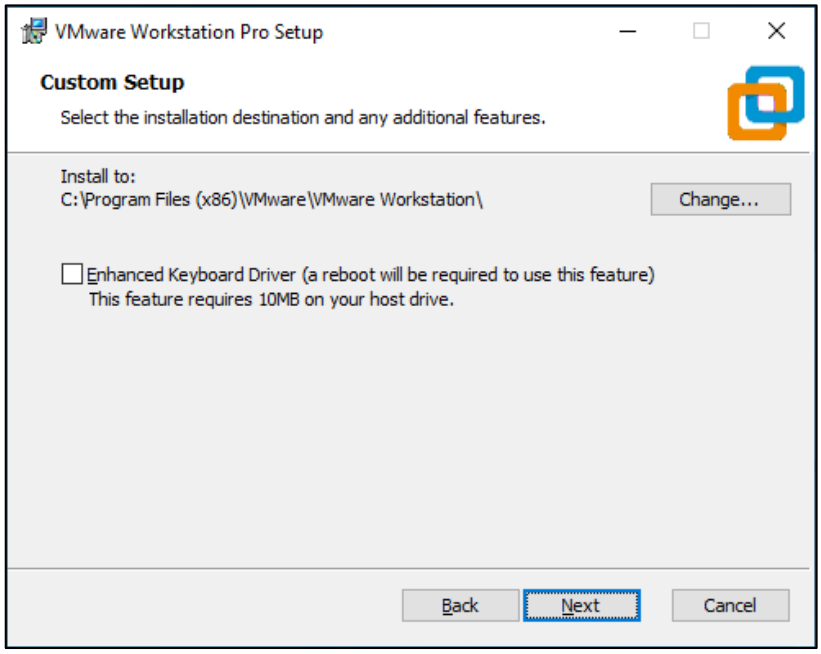
Log on to the Microsoft Windows host as an administrator user. If installing from a CD, choose Run from the Start menu and enter D:\setup.exe, where D: is the drive letter for the CD-ROM drive. If you are installing from a downloaded file, choose Run from the Start menu, browse to the directory where you saved the downloaded installer file, and run the installer. (The filename is VMwareWorkstation.exe). Click **Next** to dismiss the Welcome dialog box.



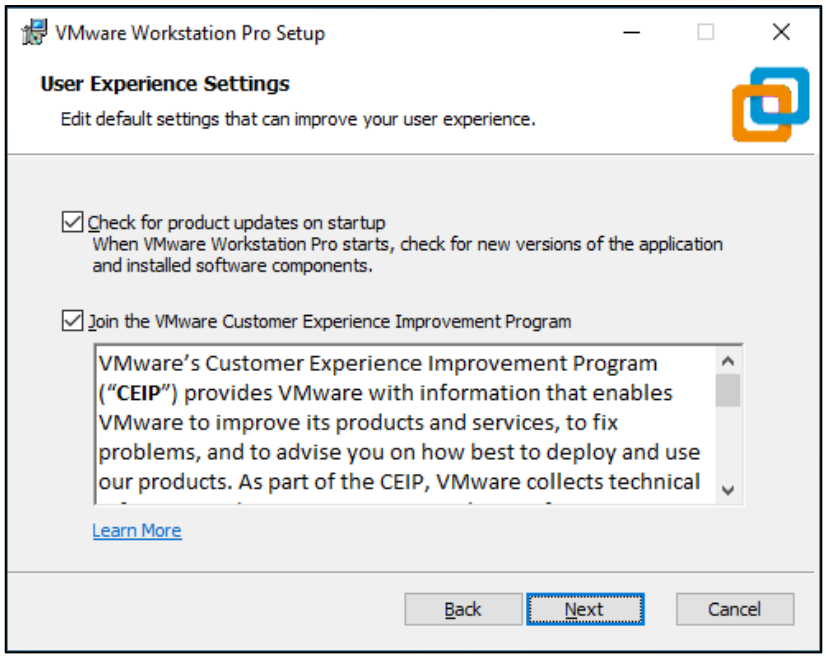
Acknowledge the end-user license agreement. Select **I Accept the terms in the License Agreement** option, then click **Next**.



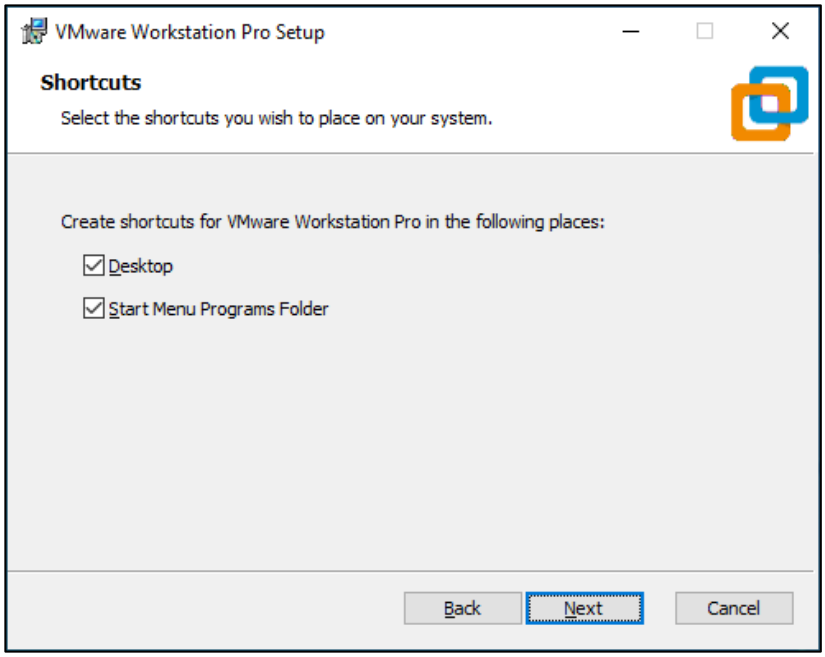
Choose the directory in which to install VMware Workstation.



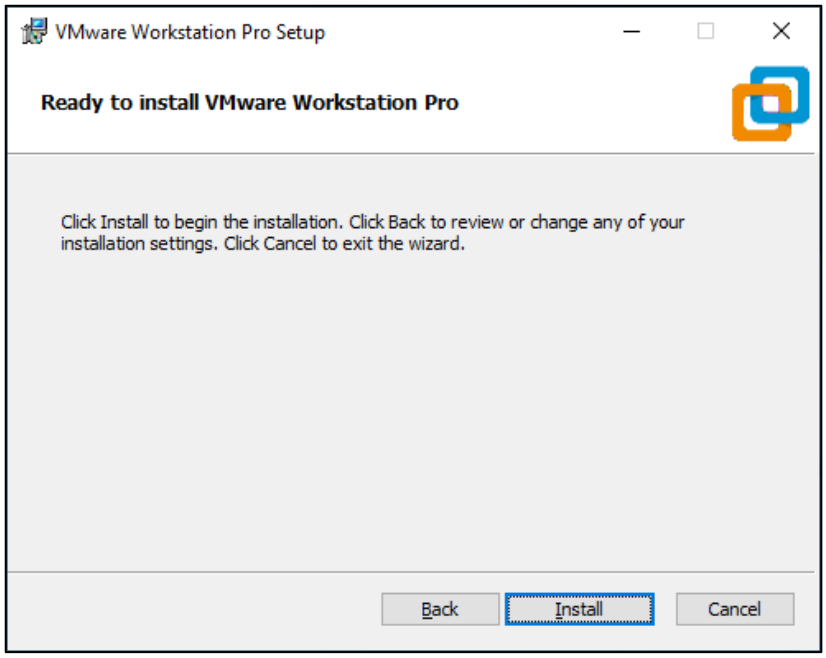
To install in another directory, then the default, click **Change** and browse to the directory of choice. If the directory does not exist, the installer creates it. Click **Next**. **Do not install VMware Workstation on a network drive.** User Experience Settings. Select *Check for updates* and press **Next**.



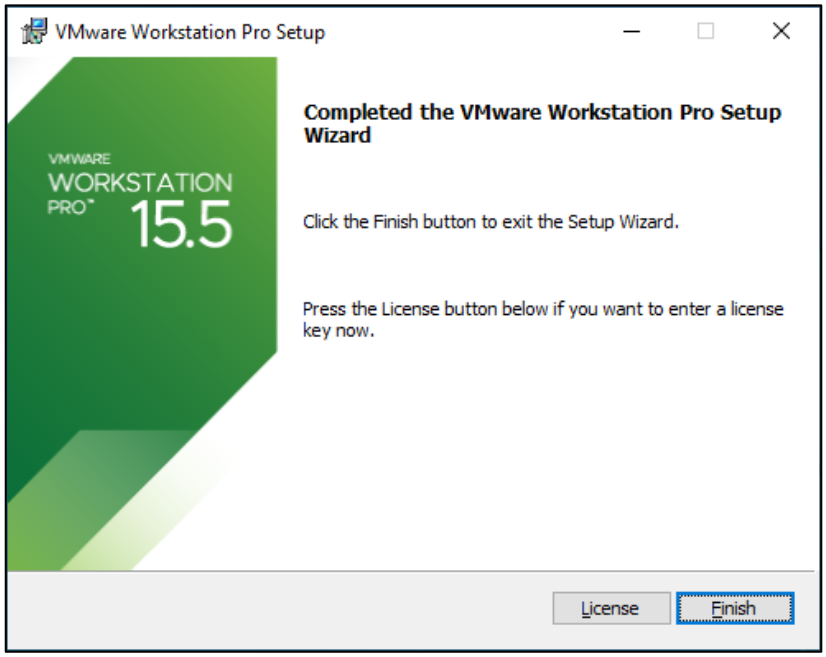
Application Shortcuts preference: select where to place the shortcuts on the system.



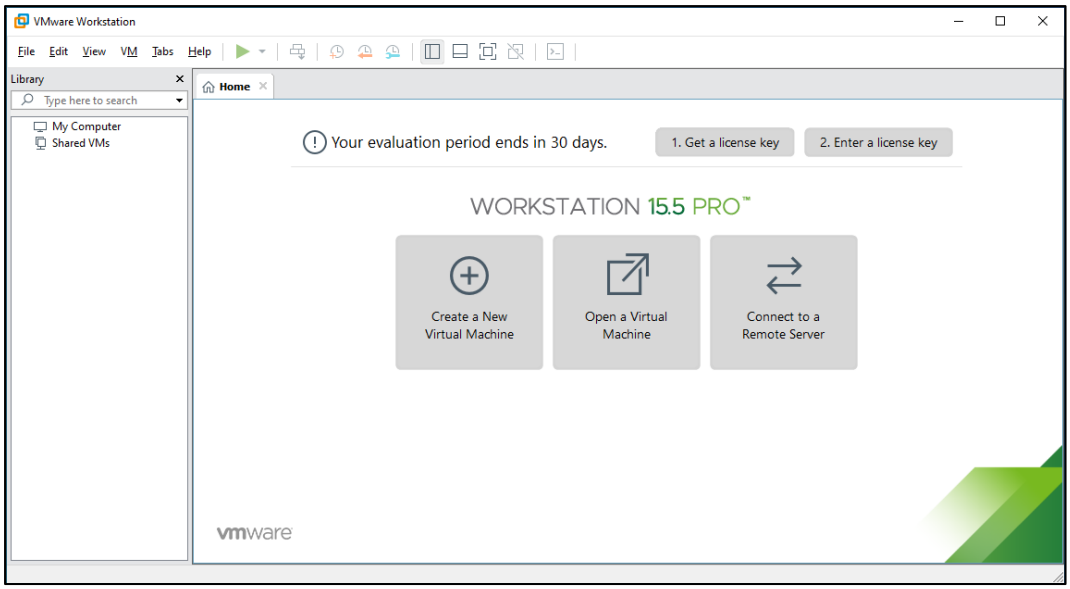
Click **Install** to begin the installation.



Click **Finish** to complete the installation process.

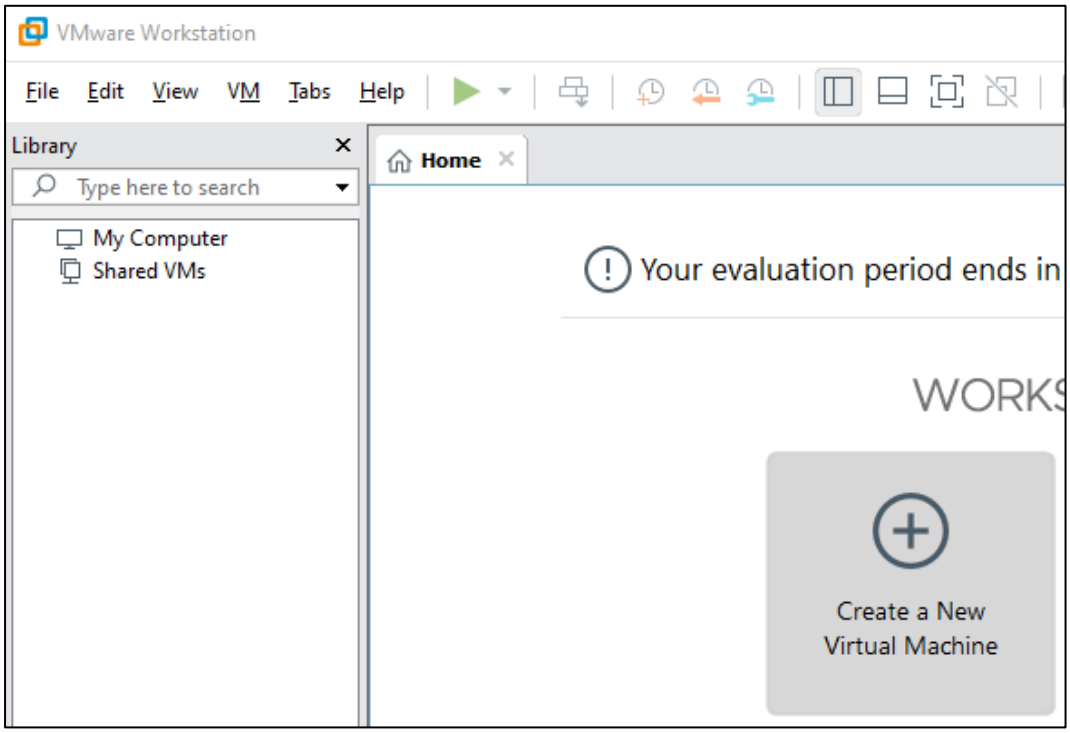


Launch VMware Workstation, double click on the icon on the desktop to launch the application.



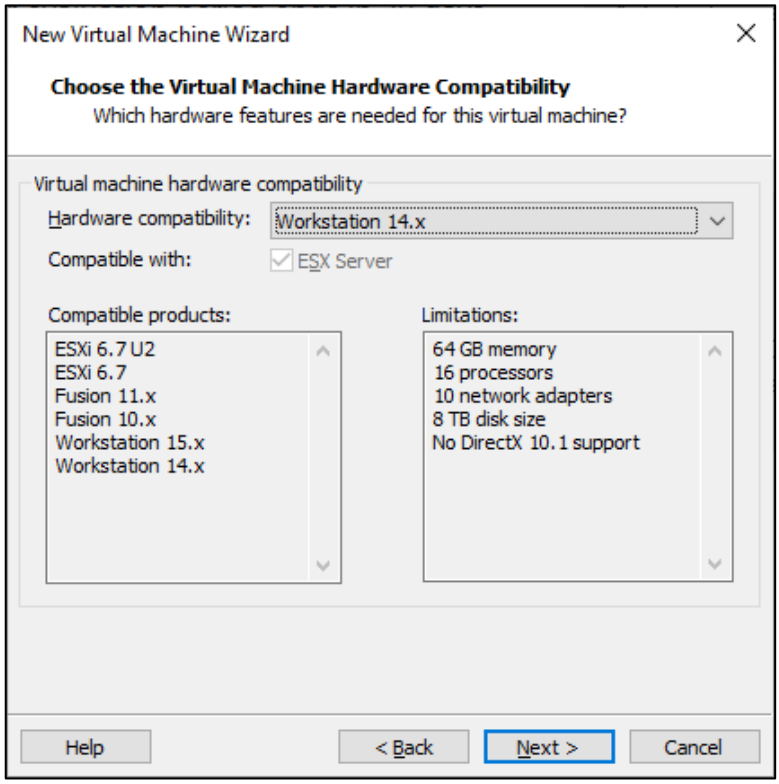
Installing Kali Linux

Click **file -> New virtual machine** or **Create a New Virtual Machine**.

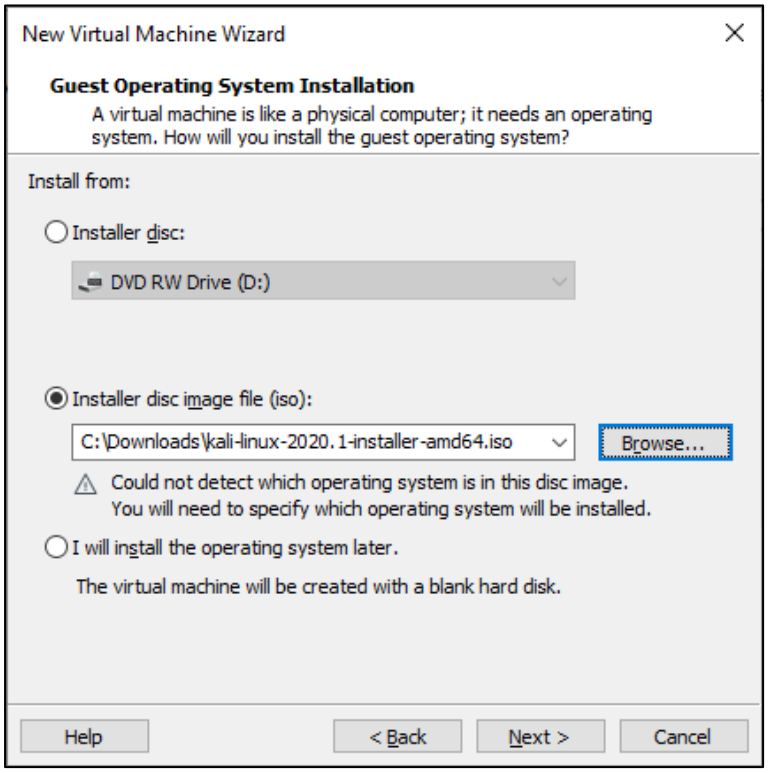


Select **Custom**.

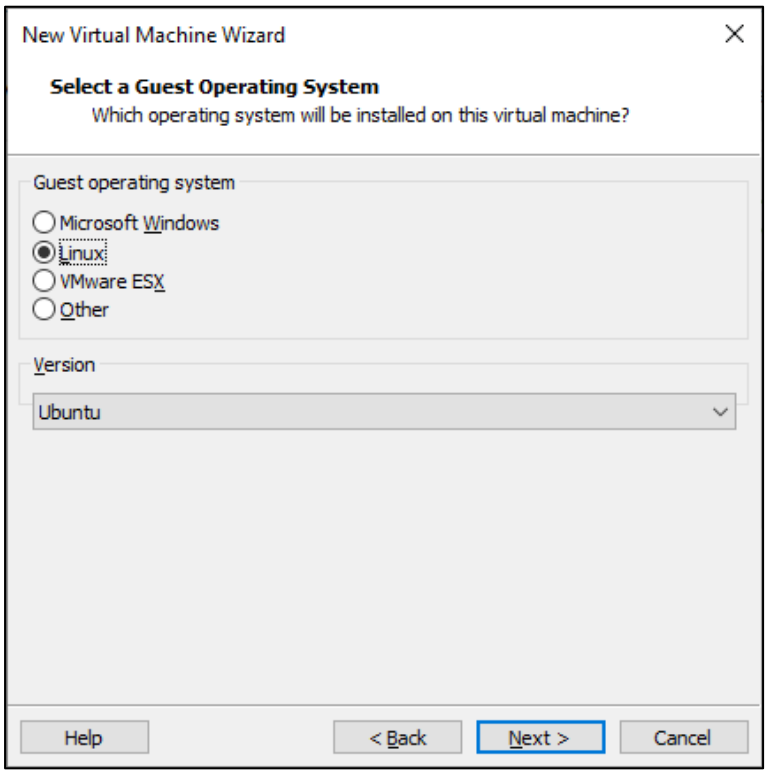
Choose the virtual machine hardware compatibility.



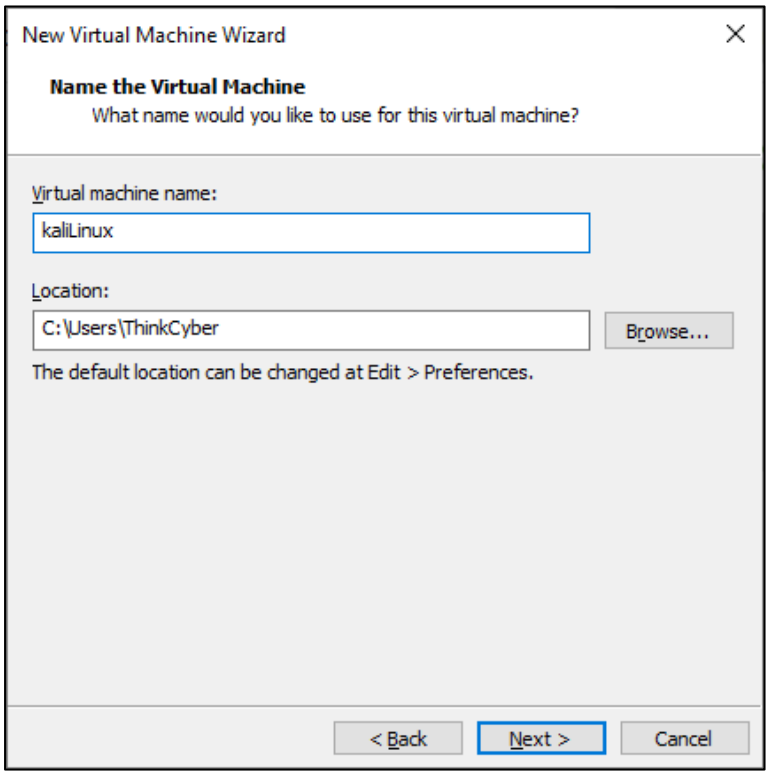
Select the installer disk image file for operating system installation, then click **Next**.



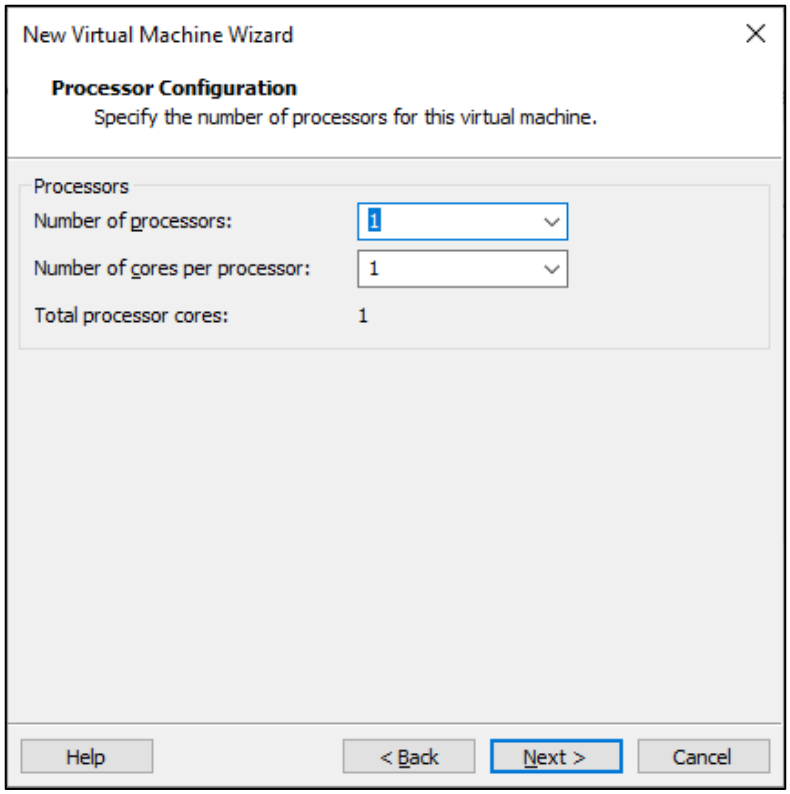
Select **Linux** as a Guest operating system. Click **Next**.



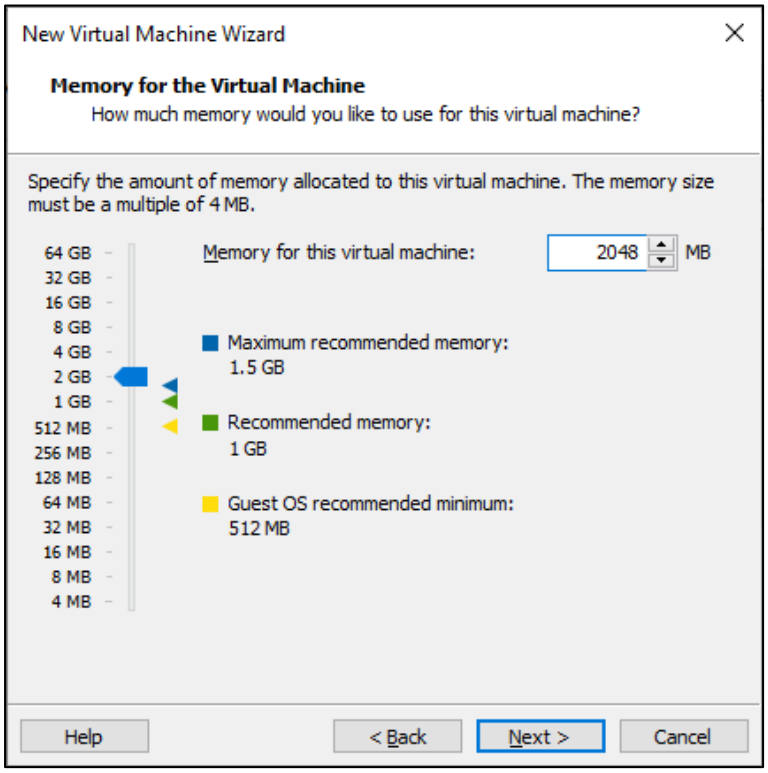
Specify the VM name and location.



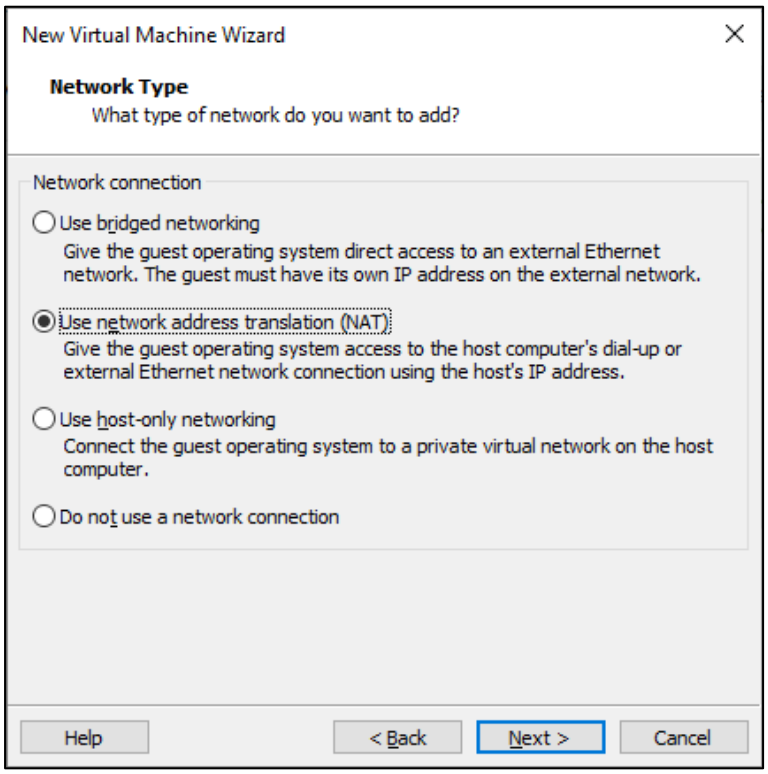
Specify the number of processors and the number of cores per processor for this virtual machine. Using one CPU is enough for Kali Linux.



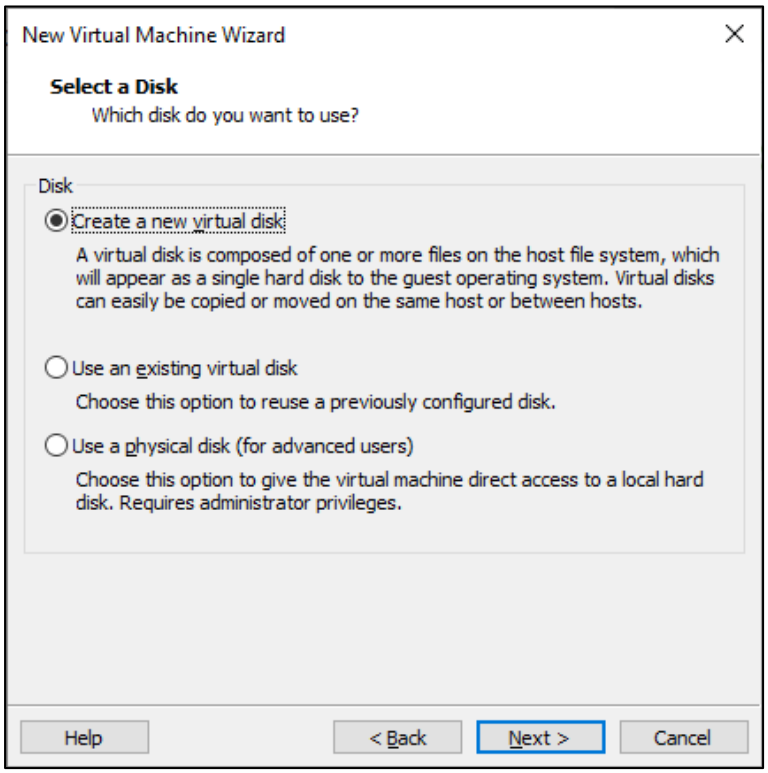
Set the Kali Linux with 2GB of memory.



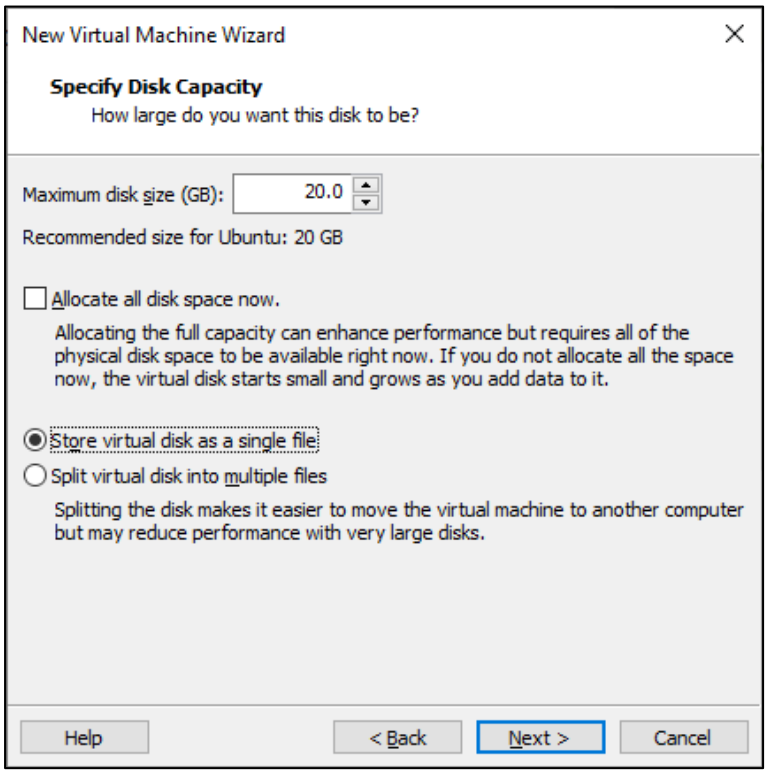
Select Network type; choose NAT.



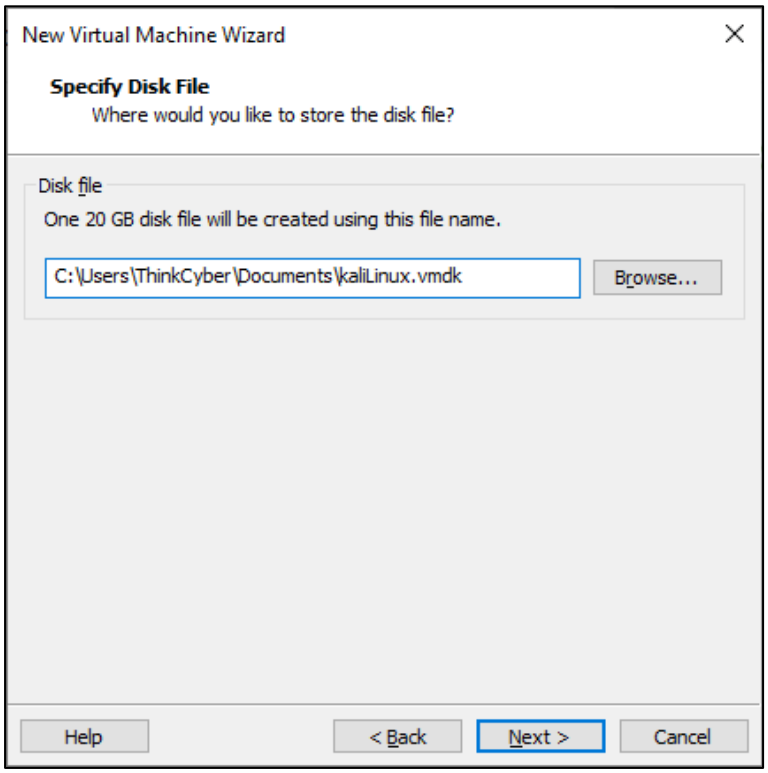
Select a disk. Click **Create a new virtual disk** and then click **Next**.



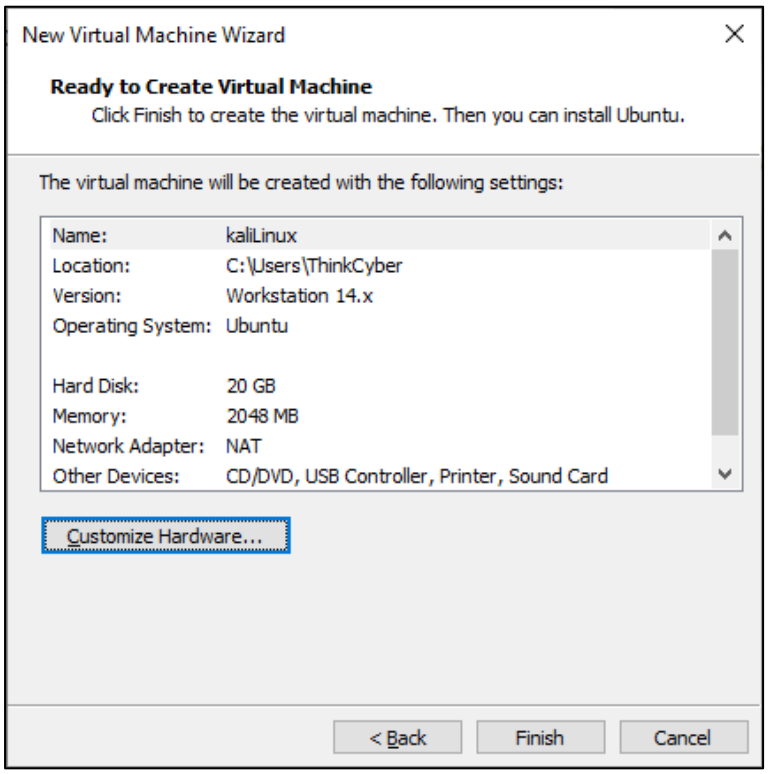
Select **Store virtual disk as a single file** if there are no limitations to the file system. Don't check the box **Allocate all disk space now** if you don't want the disk to consume all provisioned disk space.



Specify where you want to store the virtual disk file, and click **Next**.



Select **Customize hardware** if necessary, and click **Finish** to create the VM.



Introduction to Linux

Learning to work with Linux is an important skill to master. This chapter will go on a profound journey of understanding and managing the filesystem, users, and administration, learning to work with the Terminal, configuring and installing packages, and writing scripts with variables. Also, loops and commands to manipulate files and text logs, learn about data streams and use them for the purpose.

Terminal

The **Terminal** is a window you get when you open the command line interpreter. Inside the Terminal, you have the shell related to the specific language it supports. In Kali Linux, the Terminal supports Bash, python, and others.

Shell

The **shell** is the command interpreter in an operating system such as **Unix or GNU/Linux**; it is a program that executes other programs. When the shell has finished running a program, it sends an output to the screen's user, the standard output device.

The Terminal-Emulator Concept

A terminal emulator is a tool that emulates a shell while running commands and launching tools from the graphical environment. Each distribution has its way of opening the Terminal Emulator.

Linux Directories

/ - That is the root folder of the system. Everything on Linux is located under that directory.
/bin - This directory stores Linux commands (such as ping, ls, cp, and more) used by all users.
/dev - Driver, hardware, and system files.
/var - This directory contains files that are predicted to change in size and content while the systems run, like log files, for example, /var/log.
/etc - This directory contains different settings/configurations of the OS.
/lib - Directory containing libraries (shared code between applications so they could run) for critical software from **/bin** folder.
/boot - This directory contains files needed for the system activation.
/tmp - Directory containing temporary files; these files get deleted once the system is turned off and restarted; in general, they disappear after a few days.
/usr - Directory containing applications and information for users to access and operate.
/home - The personal folder of each user.
/srv - This directory contains data for system-provided services.

Linux Commands

ls	Displays the folder content. The command **ls -a** will also show hidden files (with '.' for the first character). The command **ls -l** will show each file's information, such as size, permissions, and a line for each file.
cd	Change folder.
cd ..	One folder back.
cd /	Move to the root folder.
cd ~	Move to the home folder of the user.
history	Command history of the user.
passwd	Change user password.
touch	Creates an empty file.
nano	File editor, in case the file doesn't exist - the command will create it.
cat Filename	Displays file content.
cp A B	Copies file A to location B.
mv A B	Moves file A to location B. It can also be used to rename the file.
clear	Clears the terminal screen.
pwd	Prints the full pathname of the current working directory.
echo "hello"	Creates output 'hello'.
rm -rf A	Removes folder A.
locate	Search a file in the database, and make sure to update the index beforehand.
reboot	Restart the OS without confirmation or warning.
poweroff	Shut down the OS without confirmation or warning.
man A	Extended guide for the "A" command.
uptime	Shows the overall time the system is on.
whoami	Shows the currently connected user.
sort A	Displays sorted lines in file "A" alphabetically.
head	Display the first ten lines of the file.
tail	Displays the last 10 lines of a file.
nl	Displays file content with numbered lines.
ping	Same as in Windows used to check the communication between computers.
netstat -tapn	Displays information on the active connection on the computer.
ifconfig	Displays the local net card details, including the internal IP address.
chmod	Changes permissions for the file to grant full permissions to all users and all files in a directory we are in; type: **chmod 777 ***
grep	Displays lines where the desired text is located.

File Commands

cp - used to copy files and directories.

```
root@kali: ~kali/Desktop/New/Share
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# ls
basic.exe  Share

root@kali:~kali/Desktop/New# cp basic.exe Share/

root@kali:~kali/Desktop/New# cd ./Share

root@kali:~kali/Desktop/New/Share# ls
basic.exe
```

Change the name of the file using cp.

```
root@kali: ~kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# cp basic.exe Newbasic.exe

root@kali:~kali/Desktop/New# ls
basic.exe  Newbasic.exe  Share
```

```
root@kali: ~kali/Desktop/New/Newfolder
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# cp *.exe Newfolder/

root@kali:~kali/Desktop/New# cd ./Newfolder

root@kali:~kali/Desktop/New/Newfolder# ls
basic.exe  Newbasic.exe
```

mv - used to move files from one location to another.

```
root@kali: ~kali/Desktop/Linux
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New/Newfolder# ls
basic.exe  Newbasic.exe

root@kali:~kali/Desktop/New/Newfolder# mv basic.exe /home/kali/Desktop/Linux

root@kali:~kali/Desktop/New/Newfolder# cd /home/kali/Desktop/Linux

root@kali:~kali/Desktop/Linux# ls
basic.exe
```

rm - deletes files.

```
root@kali: ~kali/Desktop/Linux
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/Linux# ls
basic.exe

root@kali:~kali/Desktop/Linux# rm basic.exe

root@kali:~kali/Desktop/Linux# ls

root@kali:~kali/Desktop/Linux#
```

cd - traversing to a specified directory.

```
root@kali: ~kali/Desktop
File  Actions  Edit  View  Help
root@kali:/opt# cd /home/kali/Desktop

root@kali:~kali/Desktop#
```

touch - creates an empty file.

```
root@kali: ~kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# ls
basic.exe  Newbasic.exe  Newfolder  Share

root@kali:~kali/Desktop/New# touch newfile.txt

root@kali:~kali/Desktop/New# ls
basic.exe  Newbasic.exe  newfile.txt  Newfolder  Share
```

pwd - prints the current directory.

```
root@kali: ~kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# pwd
/home/kali/Desktop/New
```

ls - lists all files and directories in the current location.

```
root@kali: ~kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# ls
basic.exe  Newbasic.exe  newfile.txt  Newfolder  Share
```

mkdir - creates a directory.

```
root@kali: ~kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# mkdir Newdirectory

root@kali:~kali/Desktop/New# ls
basic.exe  Newbasic.exe  Newdirectory  newfile.txt  Newfolder  Share
```

lsof - lists all recent files opened by the system.

```
root@kali: ~kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# lsof | head -5
lsof: WARNING: can't stat() fuse.gvfsd-fuse file system /run/user/1000/gvfs
      Output information may be incomplete.
COMMAND      PID    TID TASKCMD         USER   FD      TYPE             DEVICE SIZE/OFF
    NODE NAME
systemd        1                        root  cwd       DIR               8,1    36864
       2 /
systemd        1                        root  rtd       DIR               8,1    36864
```

You often need to view files or portions of them at the Linux command line. Besides, you may need to employ tools that allow you to gather data chunks or file statistics for troubleshooting or analysis purposes. The utilities in this section can assist in all these activities.

Operators in Linux

>	Saves output into a file and deletes current content if it exists.
>>	Adds output to the end of the file, for example, **echo "hello">> text.txt**
&&	Executes the first command. If successful, executes 2nd command.
;	Executes the first command and, in any case, executes 2nd command.
|	Pipeline - Afterwards, will follow commands executed on the original output before it.

Commands in APT package (advanced Package Tool)

apt-get update	Updates install packages from configured servers.
apt-get upgrade	Updates the installed packages.
apt-get dist-upgrade	Performs system updates.
apt-get install PACKAGE	Install package.
apt-get remove PACKAGE	Removes package.
apt-cache show PACKAGE	Displays the description of a package.

Before using **apt-get**, a download source must be set up. Otherwise, the system won't know where to get files from otherwise.

If needed, edit the file **/etc/apt/sources.list** (this is the link the OS goes to get files and updates) and add relevant sources.

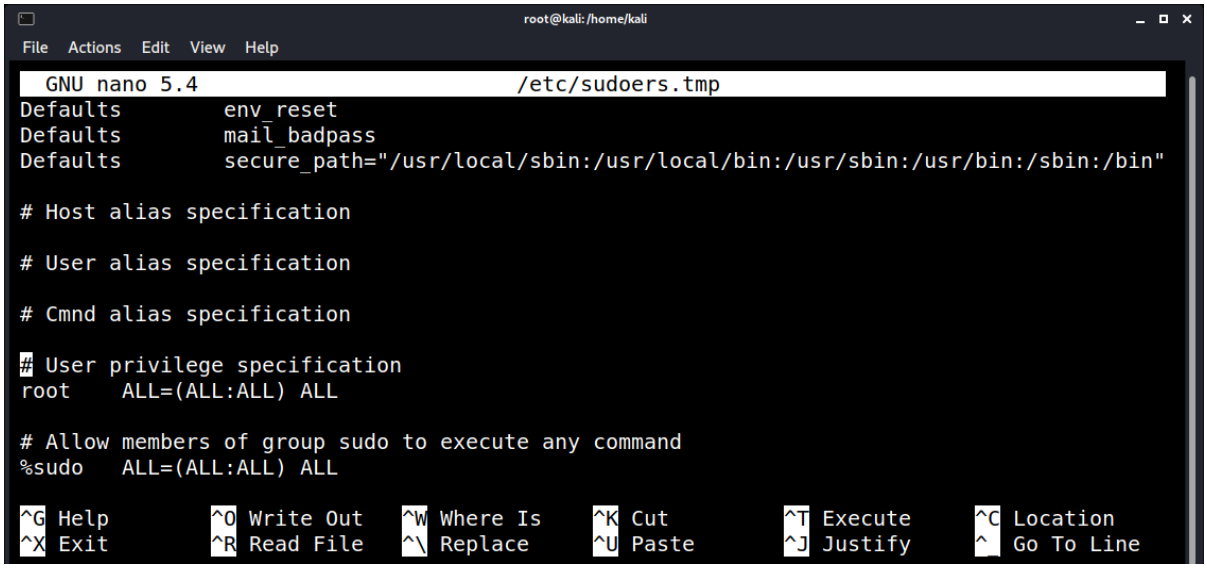
Each package in Linux has a link saved in the index. When we update Linux, more links and existing packages will also be updated.

The **/etc/apt/sources.list** file contains Linux sources. For every update, the system goes to these links and updates. It is, therefore, important to check that the sources are up to date. To check if the sources are OK, google "kali linux sources" and make sure the content is also set up in the **sources.list** file.

Linux Users

Linux is a multiuser operating system. A typical administration task in a multiuser environment is creating new users, modifying existing users, or removing users. For ease of access management, users are assigned to groups. Creating, deleting, and changing groups is also another common administration task.

In a typical **Linux** system, some users aren't allowed to execute all commands. For that, we have the **sudo** command, which allows for full permission and privileges in a specific and temporary manner. The **root** is the Admin/Superuser full privileges account that does not require the **sudo** command to execute administrator-only commands, such as the reboot command. The **sudoers** (in /etc/sudoers) contain users who can use the command **sudo** for special permission. Also, the sudo packages come with an automatic tool for editing and testing the sudoers file; the commands are: **visudo**



Understanding the Sudoers File Configurations

- **Defaults env_reset** - Resets the terminal environment after switching to root
- **root ALL=(ALL) ALL** - Allows root to do everything on any machine as any user.
- **%admin ALL=(ALL) ALL** - Allows anybody in the admin group to run anything as any user

Passwd File

The **passwd** file is located at /etc/passwd/. The file is a text file containing the attributes of each user or account on a Linux computer. The permissions for /etc/passwd are by default setting so that it is world-readable, that is so that any user on the system can read it. The file can easily be read using a text editor or with a command such as cat, which is commonly used to read files, i.e., The /etc/passwd contains one entry per line for each user (user account) system. All fields are separated by a colon (:) symbol.

```
root@kali:/home/kali
File  Actions  Edit  View  Help
root@kali:/home/kali# cat /etc/passwd | grep kali
kali:x:1000:1000:Kali,,,:/home/kali:/usr/bin/bash
```

1. **Username**: The username for login should be between 1 and 32 characters in length.
   In this case, the username is root.

2. **Password**: An x character indicates that the encrypted password is stored in the /etc/shadow file.
   In this case, The password is stored in the shadow file.

3. **User ID (UID)**: Each user must be assigned a user ID (UID). UID 0 (zero) is reserved for root, and UIDs 1-99 are reserved for other predefined accounts. Further, UID 100-999 is reserved for administrative and system accounts/groups.
   In this case, Since the user is the root, the UID is 0.

4. **Group ID (GID)**: The primary group ID (stored in /etc/group file) is the same as the UID; the GID 0 is reserved for the root group.
   In this case, The group is 0.
5. **User ID Info**: The comment field adds extra information about the users, such as the user's full name, phone number, etc.

6. **Home directory**: The absolute path to the user's directory when they log in. If this directory does not exist, then the user's directory becomes.
   In this case, The user's home folder is /root.

7. **Command/shell**: The absolute path of a command or shell (/bin/bash). Typically, this is a shell.
   In this case, The user uses the fish shell.

Shadow File

The /etc/shadow file stores the actual password in an encrypted format (more like the password's hash) for the user's account with additional user password properties. All fields are separated by a colon (:) symbol. Each user listed in the /etc/passwd file contains one entry per line.

```
root@kali:/home/kali
File  Actions  Edit  View  Help
root@kali:/home/kali# cat /etc/shadow | grep root
root:$6$i0F1s90gT0U/K8cU$yCAzp29jmmZMmwT2gXmiqxeoA21p6MnapXAErlNsMl4GA8I2b6sdKEUoFrpzbIAv/
uuvyuip5Y.YyAtqpUGb10:18934:0:99999:7:::
```

1. **Username**: The login username.
2. **Password**: The encrypted password. The password can include special characters, digits, lowercase alphabetic, and more. Usually, the password format is set to $type$salt$hashed. The types that are used on GNU/Linux are as follows:
   a. **1** is MD5
   b. **$2a$** is Blowfish
   c. **$2y$** is Blowfish
   d. **5** is SHA-256
   e. **6** is SHA-512
3. **Last password change (last changed)**: Days since Jan 1, 1970, that password was last changed.
4. **Minimum**: The minimum number of days required between password changes, i.e., the number of days left before the user can change their password.
5. **Maximum**: The maximum number of days the password is valid (after that user is forced to change their password).
6. **Warn:** The number of days before the password expires that the user is warned that their password must be changed.
7. **Inactive**: The number of days after a password expires that the account is disabled.
8. **Expire**: days since Jan 1, 1970, that account is disabled.

Text Manipulation

A filter is a program that reads standard input, operates it, and writes the results to standard output.

Grep

The *grep* command is a UNIX command utility that can find specific patterns.

cat /etc/passwd | grep kali

```
root@kali:~kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# cat /etc/passwd | grep kali
kali:x:1000:1000:Kali,,,:/home/kali:/usr/bin/bash
```

Grep Command-line Flags/Options

Furthermore, the *grep* command has a few key **flags/options**. The *--color* option. By using this option, the successful matches highlight—the case-insensitive flag. Specify the -I flag to a case-insensitive match.

```
root@kali:~kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# cat /etc/passwd | grep -i KALI --color
kali:x:1000:1000:Kali,,,:/home/kali:/usr/bin/bash
```

The "before" and "after" flags. By default, the *grep* command will show us the line with the successful match. If we want to see the lines before or after, we could use the "-A, -B, -C" flags.

-A Will print a set number of lines **after** the match.

```
root@kali:~kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# cat /etc/passwd | grep -i KALI --color -A2
kali:x:1000:1000:Kali,,,:/home/kali:/usr/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
mj:x:1001:1001:maryjane,1,1,1,1:/home/mj:/bin/bash
```

-B Will print a set number of lines **before** the match.

```
root@kali:~kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# cat /etc/passwd | grep -i KALI --color -B1
king-phisher:x:132:140::/var/lib/king-phisher:/usr/sbin/nologin
kali:x:1000:1000:Kali,,,:/home/kali:/usr/bin/bash
```

-C Will print a set number of lines **in both directions** of the match.

```
root@kali:~kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# cat /etc/passwd | grep -i KALI --color -C1
king-phisher:x:132:140::/var/lib/king-phisher:/usr/sbin/nologin
kali:x:1000:1000:Kali,,,:/home/kali:/usr/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
```

Awk

The *awk* command breaks each line of input passed into fields. By default, a field is a string of consecutive characters delimited by whitespace, though there are options for changing this.

```
root@kali:~kali/Desktop/New# cat newfile.txt
color   number  command
blue    2       printf
red     3       echo
green   4       exit
```

```
root@kali:~kali/Desktop/New# cat newfile.txt | awk '{print $3}'
command
printf
echo
exit
```

By using this command, we filtered the third column of the file. Also, filter multiple columns.

```
root@kali:~kali/Desktop/New# cat newfile.txt | awk '{print $2,$3}'
number command
2 printf
3 echo
4 exit
```

And even add a custom delimiter between them.

```
root@kali:~kali/Desktop/New# cat newfile.txt | awk '{print $2 " + " $3}'
number + command
2 + printf
3 + echo
4 + exit
```

If the line has three words, it stores $1, $2, and $3, respectively.

Sed

The *sed* command in UNIX can function as searching, finding and replacing, insertion, or deletion.

```
root@kali: ~kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# echo "Hello World"
Hello World

root@kali:~kali/Desktop/New# echo "Hello World" | sed 's/World/Earth/'
Hello Earth
```

The "s" specifies the substitution operation. The "/" are delimiters. The "world" is the search pattern, and the "earth" is the replacement string.

Word Count

Print newline, word, and byte count for each FILE and a whole line if more than one FILE is specified.

-c print the byte counts
-m print the character counts
-l print the newline counts
-w print the word counts

Count the number of lines in the file using the **-l** flag.

```
root@kali: ~kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# cat newfile.txt | wc -l
4
```

Count the number of words using the **-w** flag.

```
root@kali: ~kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# cat newfile.txt | wc -w
12
```

Or count the amount of the bytes in the file.

```
root@kali: ~kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# cat newfile.txt | wc -c
59
```

Sort and Uniq

The UNIX commands *sort* and *uniq* to order and manipulate data in text files. The *sort* command accepts input from a text file or standard output, sorts the input by line, and outputs it. The sort command will sort the given input alphabetically and numerically, prioritizing any given number by default.

```
root@kali: ~kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# cat newfile.txt
Banana
Apple
Coconut
4
1
3
```

Running the *sort* command.

```
root@kali: ~kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# cat newfile.txt | sort

1
3
4
Apple
Banana
Coconut
```

The *uniq* command takes input and removes repeated lines. Because uniq removes identical adjacent lines, it is often used in conjunction with the *sort* command to remove non-adjacent duplicate lines. This combination will sort the input and then count the repeating occurrences.

```
root@kali: ~kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# cat newfile.txt
Banana
Apple
Coconut
4
1
3
Banana
3
Apple
1


root@kali:~kali/Desktop/New# cat newfile.txt | sort | uniq -c
      1
      2 1
      2 3
      1 4
      2 Apple
      2 Banana
      1 Coconut
```

Cut

The cut command in UNIX is a command for cutting the sections from each line of files and writing the result to standard output. The primary usage of the *cut* command is cutting input by selecting specific fields. To select a field, we use the *-f* flag.

```
root@kali: ~kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:~kali/Desktop/New# cat newfile.txt
Banana  Apple   Coconut

root@kali:~kali/Desktop/New# cat newfile.txt | cut -f2
Apple
```

Head and Tail

The *head* command outputs the first part of the files, and the *tail* command is used to output the last part of the files. By default, the head and tail commands will display the first or last ten lines from the file. Specify how many lines we want to display from the beginning of the file or the end; use the *-n* flag. You often need to view files or portions of them at the Linux command line. Besides, you may need to employ tools that allow you to gather data chunks or file statistics for troubleshooting or analysis purposes. The utilities in this section can assist in all these activities.

Text Combining Commands

Putting together small text files for viewing on the screen and comparing them is helpful. The command covered will do just that. The paste is a command that allows the merging of lines of files horizontally. Its output lines consist of each file's corresponding lines specified as an argument, separated by tabs.

```
kali@kali: ~
File  Actions  Edit  View  Help
kali@kali:~$ cat newfile.txt
Dave
Simon
James
Johnny
Sarah
Sharon
George
Jouvani
```

The cat command spits the entire text file to the screen. We have user and password files separately, and we want to display them side-by-side.

```
kali@kali: ~
File  Actions  Edit  View  Help
kali@kali:~$ paste newfile.txt pass.lst
Dave    123456
Simon   11091987
James   password1
Johnny  Password2
Sarah   Mydoghulk
Sharon  987654321
George  info123
Jouvani 654321

kali@kali:~$
```

Regex in Grep

The name grep stands for "global regular expression print". This means that grep can see if the input it receives matches a specified pattern. The command grep has the useable extended regular expressions. Use the -E flag or the egrep command (same thing) to use these extended regular expressions.

Grep - Anchor Matches

Anchors are special characters that specify where a match must occur to be valid in the line. The first ones are the "**^**" and the "**$**" anchors. The "^" anchor stands for anything starting with a particular pattern.

```
kali@kali:~$ cat newfile.txt
Dave
Simon
James
Johnny
Sarah
Sharon
George
Jouvani
```

Use the syntax to filter each string that begins with the letter "J".

```
kali@kali:~$ cat newfile.txt | egrep "^J"
James
Johnny
Jouvani
```

In contrast, filter any string that ends with the letter "n" using the "**$**" anchor.

```
kali@kali:~$ cat newfile.txt | egrep "n$"
Simon
Sharon
```

Another useful anchor is the "*", which means repeating the previous character or expression zero or more times.

```
kali@kali:~$ cat fruits
apple
appppppple
apppppppppppppple
bannnnnnna
bannnaaanna
bananana
peach
peeeeeeeeach
ppppeeeeeeeaaaaccccchhhh
```

To filter any string with zero or more of the string "ap" and then "le".

```
kali@kali:~$ cat fruits | egrep "ap*le"
apple
appppppple
appppppppppppppple

kali@kali:~$
```

Grep - Grouping

Placing a group of characters within brackets specifies that the character can be anyone character found within the bracket group. The first method is "[abc]" - meaning that any **single** character will be filtered.

```
kali@kali:~$ cat fruits | grep [abn]
apple
appppppple
appppppppppppppple
bannnnnnna
bannnaaanna
bananana
peach
peeeeeeeeach
ppppeeeeeeeaaaaccchhhh

kali@kali:~$
```

The second method is "[a-d]" - which will filter any character in a range.

```
kali@kali:~$ cat fruits | grep [a-d]
apple
appppppple
appppppppppppppple
bannnnnnna
bannnaaanna
bananana
peach
peeeeeeeeach
ppppeeeeeeeaaaaccchhhh

kali@kali:~$
```

Even use anchors while using grouping, for example, **^[a-c]**. For example, find any IPs in the pattern **1XX.4X.XX.XXX.**

```
kali@kali:~$ cat IP | egrep 1[0-9][0-9].4[0-9].[0-9][0-9][0-9].[0-9][0-9][0-9]
104.40.239.255
104.40.239.128

kali@kali:~$
```

Use *sort* and *uniq* commands to filter the repeating IP addresses.

```
kali@kali:~$ cat IP | egrep 1[0-9][0-9].4[0-9].[0-9][0-9][0-9].[0-9][0-9][0-9] | sort | un
iq
104.40.239.128
104.40.239.255

kali@kali:~$
```

Grep - Times

This regex is used to find a match that repeats more than once. The first one is "\{n\}", which is used to filter strings that repeat "n" times exactly.

```
kali@kali:~$ cat IP | egrep 1\{2}
192.99.200.113
99.86.117.182
192.99.200.113
99.86.117.182

kali@kali:~$
```

See that the *grep* command matched the number "1" repeated twice. Furthermore, use the expression "\{n,m\}", which will match any string from n to m times.

```
kali@kali:~$ cat IP | egrep 1\{1,2}
192.168.221.128
104.40.239.255
104.40.239.128
192.99.200.113
99.86.117.182
192.99.200.113
```

The *grep* command matched any time the number "1" appeared once or twice. The last usage of "times" in grep is when we need the *grep* command to match at least n times; use the syntax:

"\{n,\}".

```
kali@kali:~$ cat IP | egrep 2\{1,}
192.168.221.128
104.40.239.255
104.40.239.128
192.99.200.113
99.86.117.182
192.99.200.113
```

The *grep* command matched any occurrence of the number "2".

Grep - Special Expressions

In grep, the '\' character (backslash) takes a special meaning when followed by certain ordinary characters.

"\s"	White Space
"\S"	non-White Space
"\d"	digit character
"\D"	non-digit character
"\w"	Word
"\W"	non-Word (punctuation, spaces)

Regex in Awk, Sed, and Cut

Like the *grep* command, a few other commands have a unique Extended Regular Expression.

```
kali@kali:~$ cat IP | awk '/192.99.200.113/{print $0}' | uniq
192.99.200.113

kali@kali:~$
```

Like the *grep* command, the *awk* command can filter a specific pattern; what is unique is specifying a particular column. It is worth mentioning that the command *sed* is also capable of pattern matching.

```
kali@kali:~$ cat IP | sed -n  '/192.99.200.113/p'  | uniq
192.99.200.113

kali@kali:~$
```

Like the *grep* command anchors, the same characters can be used in the commands *awk* and *sed*. For example, filter any IP address starting with "192".

```
kali@kali:~$ cat IP | awk '/^192/{print $0}' | sort -n | uniq -c
      1 192.168.221.128
      2 192.99.200.113
```

Or any IP address ending with the number "3".

```
kali@kali:~$ cat IP | awk '/3$/{print $0}' | sort -n | uniq -c
      1 9.86.117.183
      2 192.99.200.113
```

Streams, Redirection, and Pipes

Data streams are the raw materials that command-line tools and Linux utilities use to receive and send data.

Three Types of Streams

Stdin (0) - this is the stream that programs use to read input data. For example, the commands *dir* and *ls* can use command-line arguments, but they work without stream data input.

Stdout (1) - this is the stream to which programs output their data. The best example is the command *cat*. When you use a file, it will print the files in contact with the user's screen to see.

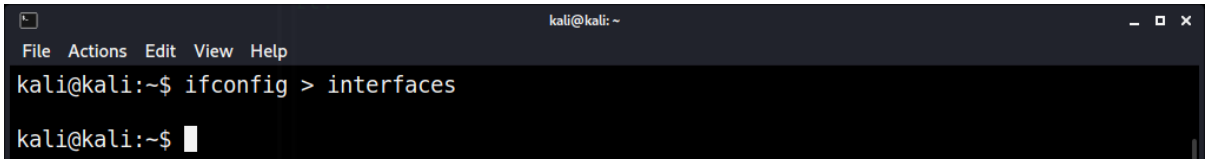
Stderr (2) - these are the stream programs used for errors. It's also printed on the screen like stdout for diagnostics and troubleshooting.

Redirection

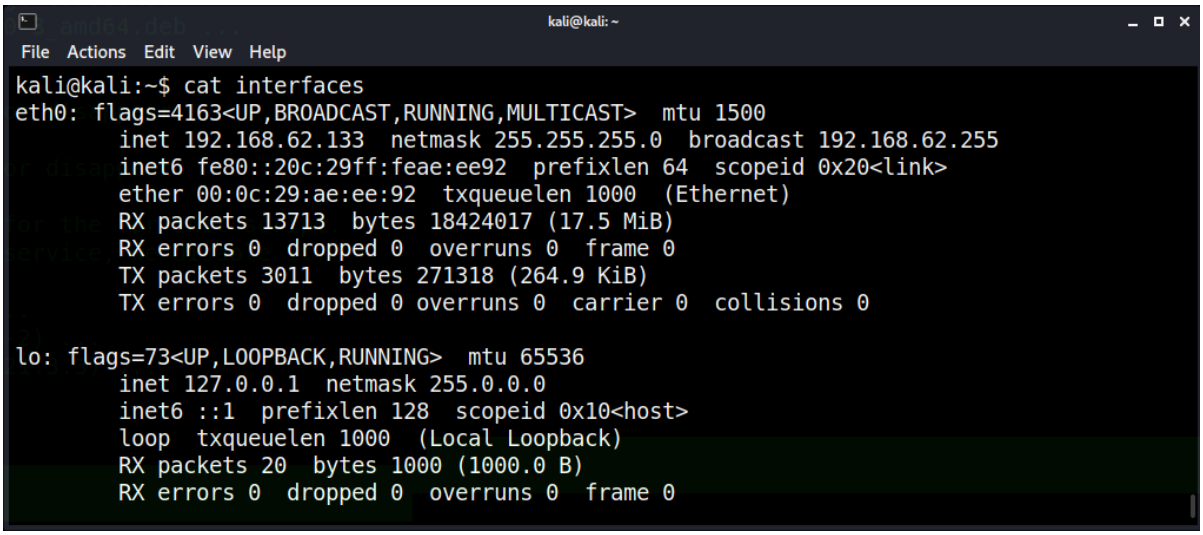
The redirect capabilities give you a handy toolbox to accomplish tasks faster and improve productivity. Redirect any of the data streams.

> overwrite
>> append

Save the command output into a text file using the **>** character.



Use the **cat** to see the file output.



What will happen when I do **>** again?

```
kali@kali:~$ echo "192.168.221.130" > interfaces

kali@kali:~$ cat interfaces
192.168.221.130
```

The file contents were replaced with the output of the echo I wrote. Suppose I want to append text to a file; use **>>**. The new IP address is added to the last line of the file.

```
kali@kali:~$ echo "192.168.221.137" >> interfaces

kali@kali:~$ cat interfaces
192.168.221.130
192.168.221.137
```

Pipes

A pipe is a form of redirection used in Linux and other Unix-like operating systems to send the output of one command/program/process to another command/program/process for further processing. For example, read a file's content and then utilize one of the text manipulation tools we learned before.

The Linux pipe is the character: |

```
kali@kali:~$ cat IP
192.168.221.128
104.40.239.255
104.40.239.128
192.99.200.113
99.86.117.182
192.99.200.113
99.86.117.182
9.86.117.183
```

What if we want to search for IP addresses starting with 192. And see that on the screen?

```
kali@kali:~$ cat IP | grep "^192"
192.168.221.128
192.99.200.113
192.99.200.113
```

The difference between | and || is:
| - (Condition 1 | Condition 2): checks both cases even if case 1 is true
|| - (Condition 1 || Condition 2): doesn't bother to check the second case if the first one is true.

Searching in Linux

Linux, being a powerful and flexible operating system, offers a variety of tools to search for files and directories. Two of the most used tools for this purpose are **find** and **locate**. In this article, we'll explore the intricacies of these tools, their differences, and how to use them effectively.

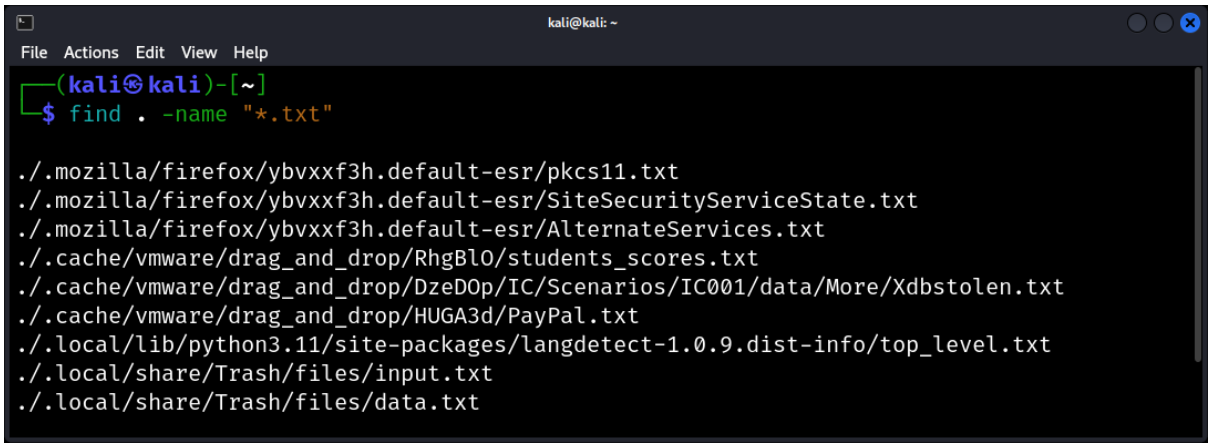
The find Command

The **find** command is a versatile tool that allows users to search for files and directories based on various criteria such as name, type, size, and more. It searches the directory tree rooted at each given file name by evaluating the given expression from left to right, according to the rules of precedence, until the outcome is known.

Basic Usage

The basic syntax of the find command is: **find [path...] [expression]**

For example, to find all files named all the text files in the current directory and its subdirectories:



Commonly Used Options

- **-name**: Search for files based on their name.
- **-type**: Specify the type of file (e.g., **f** for regular files, **d** for directories).
- **-size**: Search for files based on their size.
- **-mtime**: Search for files modified within a specified number of days.
- **-user**: Search for files owned by a specific user.

Examples

1. **Find all directories named "docs"**:

   find / -type d -name "docs"

2. **Find all files larger than 100MB**:

   find / -type f -size +100M

3. **Find all files modified in the last 7 days**:

   find / -type f -mtime -7

4. **Find all files owned by the user "john"**:

   find / -type f -user john

Combining Expressions
You can combine multiple expressions using logical operators:

- **-and**: Both the preceding and following expressions must be true (this is the default behavior).
- **-or**: Either the preceding or the following expression must be true.
- **-not**: Negates the following expression.

For example, to find all **.txt** files modified in the last 7 days but not owned by "john":

find / -type f -name "*.txt" -mtime -7 -not -user john

The locate Command
While **find** searches the filesystem in real-time, **locate** uses a prebuilt database of files and directories to provide faster search results. This database is typically updated daily using the **updatedb** command.

Basic Usage
The basic syntax of the locate command is: **locate [options] pattern**

For example, to locate all files and directories named "example.txt":

locate example.txt

Commonly Used Options

- **-i**: Ignore case distinctions.
- **-l**: Limit the number of results.
- **-b**: Match only the base name against the specified patterns.

Examples

1. **Locate all files and directories named "config" (case-insensitive)**:

   ```
   locate -i config
   ```

2. **Limit the search results to 10 entries**:

   ```
   locate -l 10 example.txt
   ```

Differences Between *find* and *locate*

1. **Speed**: **locate** is generally faster than **find** because it queries a prebuilt database. However, this means that **locate** might not always have the most up-to-date information.

2. **Real-time vs. Database**: **find** searches the filesystem in real-time, while **locate** relies on a database updated by **updatedb**.

3. **Flexibility**: **find** offers more flexibility in terms of search criteria and expressions.

Network Services in Linux

Linux, as a robust and versatile operating system, plays a pivotal role in the world of networking. From simple file sharing to complex web services, Linux offers a wide array of network services that cater to various needs. In this article, we'll delve deep into the realm of network services in Linux, exploring their significance, common services, and configuration basics.

What are Network Services?

Network services refer to applications or processes that run on a server and provide functionalities to other computers (clients) over a network. These services listen on specific ports and wait for incoming requests. Once a request is received, the service processes it and sends back the appropriate response.

Common Network Services in Linux

1. SSH (Secure Shell)

- **Purpose**: Secure remote command execution and file transfer.
- **Port**: 22
- **Key Software**: OpenSSH

SSH allows users to securely connect to a remote machine. It encrypts the session, ensuring that eavesdroppers cannot decipher the data being transmitted.

2. HTTP/HTTPS (HyperText Transfer Protocol/Secure)

- **Purpose**: Web services.
- **Port**: 80 for HTTP, 443 for HTTPS
- **Key Software**: Apache, Nginx, Lighttpd

These protocols are the backbone of the World Wide Web, serving web pages and other web content to users.

3. FTP (File Transfer Protocol)

- **Purpose**: File transfer.
- **Port**: 21
- **Key Software**: vsftpd, ProFTPD

FTP is a standard network protocol used to transfer files from one host to another over a TCP-based network.

4. DNS (Domain Name System)

- **Purpose**: Domain name resolution.
- **Port**: 53
- **Key Software**: BIND, dnsmasq

DNS translates human-friendly domain names (like **www.example.com**) into IP addresses.

5. SMTP (Simple Mail Transfer Protocol)

- **Purpose**: Email transmission.
- **Port**: 25
- **Key Software**: Postfix, Sendmail

SMTP is used to send emails between servers and from clients to servers for email submission.

6. NFS (Network File System)

- **Purpose**: File sharing over a network.
- **Port**: Varies, often 2049
- **Key Software**: nfs-utils

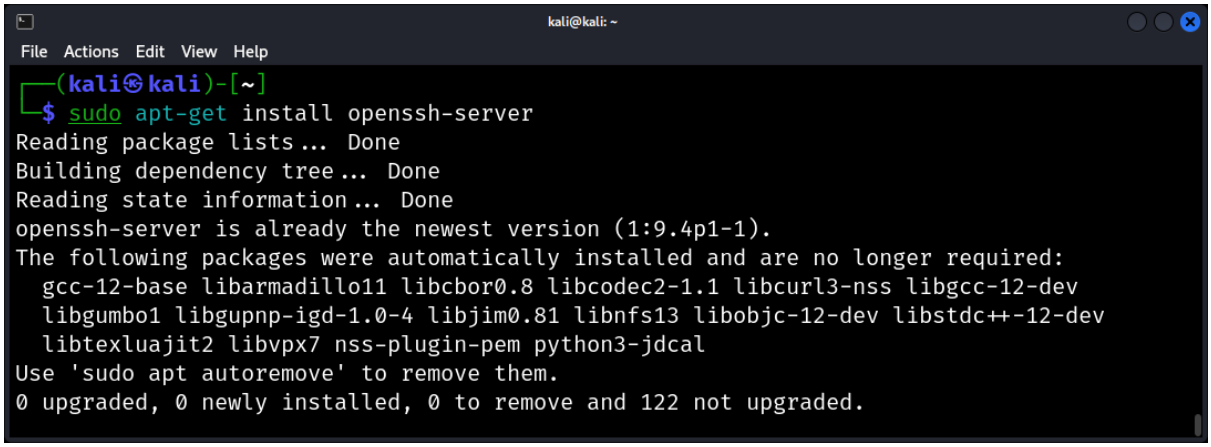
NFS allows a system to share directories and files with others over a network.

Configuring Network Services

While the exact configuration steps vary for each service, here's a general approach:

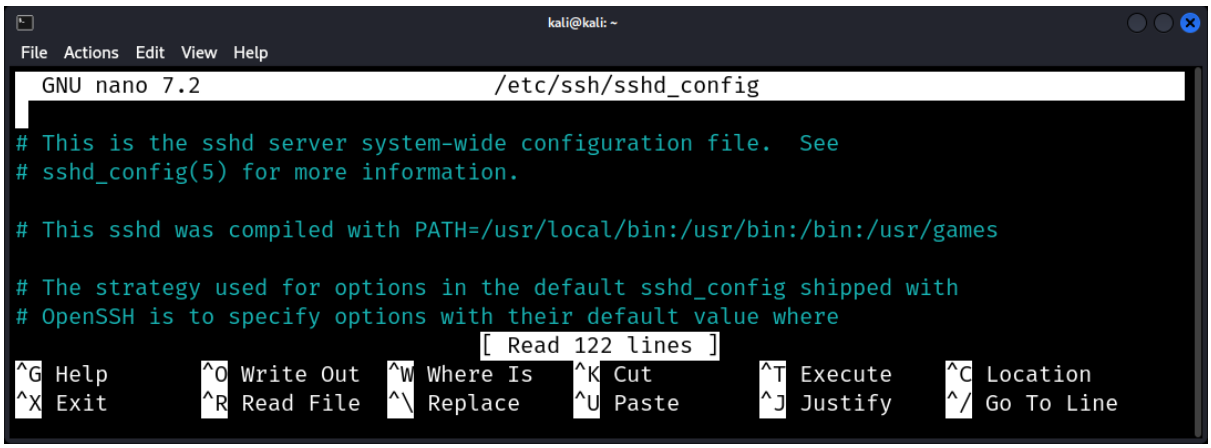
1. **Installation**: Use the package manager specific to your Linux distribution (e.g., **apt** for Debian/Ubuntu, **yum** for CentOS) to install the desired service.

   sudo apt install openssh-server



2. **Configuration**: Most services have configuration files located in **/etc/** or a subdirectory thereof. Edit these files to customize the service's behavior.

   sudo nano /etc/ssh/sshd_config



3. **Control the Service**: Use the **service** command to start, stop, or restart the service.

   sudo service ssh restart

4. **Testing**: After configuring a service, always test to ensure it's working as expected. For instance, for SSH, you can use the **ssh** command:

   ssh username@localhost

```
kali@kali: ~
File  Actions  Edit  View  Help
┌──(kali㉿kali)-[~]
└─$ sudo service ssh restart

┌──(kali㉿kali)-[~]
└─$ ssh kali@localhost
The authenticity of host 'localhost (::1)' can't be established.
ED25519 key fingerprint is SHA256:lGN9fV7hYiWVdhmDuVm35bd3EfFQrJsN7UhUaPLuqhU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'localhost' (ED25519) to the list of known hosts.
kali@localhost's password:
Linux kali 6.5.0-kali2-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.5.3-1kali2 (2023-10-03) x86
_64

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
```

Security Considerations

When running network services:

1. **Minimize Attack Surface**: Only run necessary services. If a service isn't needed, disable it.

2. **Regular Updates**: Keep all services updated to patch vulnerabilities.

3. **Use Firewalls**: Restrict access to services by using firewalls. Only allow necessary ports and IP addresses.

4. **Monitoring**: Regularly monitor logs and use tools like **netstat** or **ss** to check listening ports and established connections.

Linux Permissions

Every file and directory in the Unix/Linux system is assigned three types of permissions.

User

A user is the owner of the file. By default, the person who created a file becomes its owner. Hence, a user is also sometimes called an owner.

Group

A user- group can contain multiple users. All users belonging to a group will have the same access permissions to the file. Suppose you have a project where several people require access to a file. Instead of manually assigning permissions to each user, you could add all users to a group and assign group permission to the file. These group members and no one else can read or modify the files.

Other

Any other user who has access to a file. This person has neither created the file nor belongs to a user group that could own the file. Practically, it means everybody else. Hence, when you set permission for others, it is also referred to as set permissions for the world. To see permissions of files and information in a more detailed way, type **ls -l**

```
root@kali: /home/kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:/home/kali/Desktop/New# ls -l
total 336
-rwxrwxrwx 1 root root  38587 Nov  1 03:18  auth.log
-rwxr-xr-x 1 kali kali  16040 Oct 31 02:34  basic.exe
-rw-r--r-- 1 kali kali 284816 Nov  2 04:00 'date1%3fBNLv65=pAAS'
```

Additionally, execute the same command for a specific file using **ls -l FILENAME**.

```
root@kali: /home/kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:/home/kali/Desktop/New# ls -l basic.exe
-rwxr-xr-x 1 kali kali 16040 Oct 31 02:34 basic.exe
```

We have highlighted '**-rw-r-r--**' this code tells us about the permissions given to the owner, user group, and others. The first '-' implies that we have selected a file.

```
root@kali: /home/kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:/home/kali/Desktop/New# ls -l basic.exe
-rwxr-xr-x 1 kali kali 16040 Oct 31 02:34 basic.exe
```

Otherwise, if it were a directory, **d** would have been shown.

```
root@kali: /home/kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:/home/kali/Desktop/New# ls -l
total 4
drwxr-xr-x 2 kali kali 4096 Oct 26 09:08 share
```

- Read the file
- Write or edit the file
- He cannot execute the file since the execute bit is set to '-'

```
root@kali: /home/kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:/home/kali/Desktop/New# ls -l
total 16
-rw-r--r-- 1 kali kali 16040 Oct 31 02:34 basic.exe
```

chmod permissions filename. Use the **chmod** command, which stands for *change mode*. Set permissions (read, write, execute) on a file/directory for the owner, group, and the world using the command.

Syntax: **chmod <option> file/folder**

Each user can have different permissions for a file.

x executes
r read
w writes

The permissions are divided into numbers: 1, 2, and 4 are the base numbers of **Linux,** and from those numbers, create the permissions.

Absolute (Numeric) Mode

Permission Type	Symbol	Numeric	Number
Execute	**x**	**1**	**1**
Write	**w**	**2**	**2**
Execute + Write	x+w	1+2	3
Read	**r**	**4**	**4**
Read + Execute	r+x	4+1	5
Read + Write	r+w	4+2	6
Read + Write + Execute	r+w+x	4+2+1	7

Understanding file permissions by three-digit octal number.

```
root@kali: /home/kali/Desktop/New
File  Actions  Edit  View  Help
root@kali:/home/kali/Desktop/New# chmod 764 basic.exe

root@kali:/home/kali/Desktop/New# ls -l
total 16
-rwxrw-r-- 1 kali kali 16040 Oct 31 02:34 basic.exe
```

In the window, we have changed the permission of the file 'kalissh' to '764'.

7 Read + Write + Execute (rwx) : file owner
6 Read + Write (rw -) : user group
4 Read (r - -) : everyone else

Symbolic Mode

In the Absolute mode, you change permissions for all three owners. In the symbolic mode, modify the permissions of a specific owner.

Operator	Description
+	Adds permission to a file directory
-	Remove permission
=	Sets the permission and overrides the permissions set earlier

User Denotations	Ownership
u	user/owner
g	group
o	other
a	all

- Current file permissions.

```
root@kali:/home/kali/Desktop/New# ls -l basic.exe
---------- 1 kali kali 16040 Oct 31 02:34 basic.exe
```

- Setting permissions to the other users.

```
root@kali:/home/kali/Desktop/New# chmod o=rwx basic.exe

root@kali:/home/kali/Desktop/New# ls -l
total 16
-------rwx 1 kali kali 16040 Oct 31 02:34 basic.exe
```

- Adding 'execute' permissions to the user group.

```
root@kali:/home/kali/Desktop/New# chmod g+x basic.exe

root@kali:/home/kali/Desktop/New# ls -l
total 16
------xrwx 1 kali kali 16040 Oct 31 02:34 basic.exe
```

- Removing 'read' permissions for the 'user'.

```
root@kali:/home/kali/Desktop/New# ls -l
total 16
-r----xrwx 1 kali kali 16040 Oct 31 02:34 basic.exe
```

```
root@kali:/home/kali/Desktop/New# chmod u-r basic.exe

root@kali:/home/kali/Desktop/New# ls -l
total 16
------xrwx 1 kali kali 16040 Oct 31 02:34 basic.exe
```

Changing Ownership and Group

For changing the ownership of a file/directory, use the command **chown <user>**.

```
root@kali:/home/kali/Desktop/New# chown root basic.exe

root@kali:~kali/Desktop/New# ls -l
total 16
------xrwx 1 root kali 16040 Oct 31 02:34 basic.exe
```

To change the user and the group for a file or directory, use the command:

chown <user:group> filename

```
root@kali:~kali/Desktop/New# chown root:root basic.exe

root@kali:~kali/Desktop/New# ls -l
total 16
------xrwx 1 root root 16040 Oct 31 02:34 basic.exe
```

Bash Scripting and Automation

Bash (AKA Bourne Again Shell) is a command shell primarily in **Linux** operating systems. A shell script is a fully-fledged programming language in itself. It can define variables and functions and do conditional execution of shell commands.

Start with creating an empty file for the first bash script.

nano firstscript.sh

Use "Shebang" to fully use the shell features and ensure that the shell will interpret the commands.

#!interpreter [arguments]

The interpreter is the full path to a binary file (ex: /bin/sh, /bin/bash), and the arguments are optional. Without this line, the script will be launched via the shell from which the script was called; for example, if we wrote a script based on Bash shell features, but the user runs the script from the ksh shell, the script will run as a ksh shell script, and therefore the script will not work.

#!/bin/bash

As we learned before, any executable (runnable) file in the Linux environment must have the appropriate executable permissions, allow any user to read, write, and execute the script:

chmod firstscript.sh

Variables

A variable is a character string to which we assign a value. The value assigned could be a number, text, filename, device, or other data type. A variable is nothing more than a pointer to the actual data. The shell enables you to create, assign, and delete variables. The name of a variable can contain letters (a to z or A to Z), numbers (0 to 9), or the underscore character (_). Unlike many other programming languages, Bash does not segregate its variables by "type"; therefore, any declared variable is considered a "character string" used according to its context; define a variable as follows:

var=”value”

To access the value stored in a variable, type the variable name prefixed with a dollar sign ($); for example, in the previous demonstration, we set a variable named **var** and assigned the value of "**value**".

echo $var

It is essential to notice that in the Bash shell, accessing variables using a plain syntax of "$var" is a simplified version of using curly braces, for example, "${var}". If they are the same, why do curly braces exist? Well, the "curl braces" can do things the "simplified" version can't, for instance, reference an array index or remove a substring.

The Declare Command

The declare command is used to set variable values and attributes. By default, all variables are set as a string and as readable.

declare [flag] [variable]

To enable the attribute, use a "-" addition to the flag, and to disable it, use the "+" sign. The command has a lot of functions.

Flag		Description
-r	+r	Make the named items read-only. They cannot subsequently be assigned values or unset.
-i	+i	Give the named items the integer attribute. Values assigned to the variable will be restricted to integer values. If a non-integer value is assigned, an error is reported, or 0 (zero) is assigned instead.
-l	+l	Convert all uppercase letters to lowercase.
-u	+u	Convery all lowercase letters to uppercase.
-A		Declare the named items to be associative arrays. This attribute cannot be unset.
-a		Declare the named items to be indexed arrays. This attribute cannot be unset.
-p		Displays the options and attributes of each variable name.

Conditions - The IF Conditions

With the IF condition, define that if a specific condition is true, then something will happen.

```
#!/bin/bash

if [ condition ]
then
    [ command ]
fi
```

Besides, state what will happen if the condition is not true using the else command.

```
#!/bin/bash

if [ condition ]
then
    [ command1 ]
else
    [ command2 ]
fi
```

Furthermore, we could tell the interpreter to test another condition if the first condition is false using the elif statement (stands for else if).

```
#!/bin/bash

if [ condition1 ]
then
    [ command1 ]
elif [ condition2 ]
then
    [ command2 ]
fi
```

There are a lot of built-in checks and comparisons, but the handiest ones are as follows:
Syntax: if [$<var> -eq/-ne/-lt/-le/-gt/-ge <number>]

-eq	Equals
-ne	Not equals
-lt	Less then
-le	Less or equal
-gt	Greater than
-ge	Greater or equal

Syntax: if [-a/-e/-z $<var>]

-a	True if FILE exists.
-e	True if FILE exists.
-z	True if VAR is non-empty.

To test if the variable "var1" exists, and then if it does exist, test if it is bigger than five or smaller.

```
#!/bin/bash

if [ ! -z $var1 ]
then
    if [ $var1 -gr 5 ]
    then
        echo "Var1 exists and bigger than 5"
    else
        echo "Var1 exists and smaller than 5"
    fi
else
    echo "Var1 is nonexistent"
fi
```

Loops

We could use a loop when running a command or a sequence of commands several times.

For Loop

The for loop iterates over a list of items while inputting each item into a temporary variable and performing the given commands.

```
#!/bin/bash

for var in [ list ]
do
    [ command ]
done
```

Use the "seq" command to run over a list of numbers.

```
#!/bin/bash

for var in $(seq 1 5)
do
    echo "$var"
done
```

This loop will input the given number (1,2, etc.) into the temporary variable (var), and then the loop will print the variable (using echo).

```
root@kali:/home/kali# bash while.sh
1
2
3
4
5
root@kali:/home/kali#
```

Add a word to each line of a text file.

```
root@kali:/home/kali# cat file.txt
1
2
3
4

root@kali:/home/kali#
```

```
#!/bin/bash

for var in $(cat file.txt)
do
    echo "Test: $var"
done
```

```
root@kali:/home/kali# bash while.sh
Test: 1
Test: 2
Test: 3
Test: 4

root@kali:/home/kali#
```

Input a cost list into the *for* command.

```
#!/bin/bash

for var in First Second Third
do
    echo "$var"
done
```

```
root@kali:/home/kali# bash while.sh
First
Second
Third

root@kali:/home/kali#
```

While Loops

The Bash while loop is a control flow statement that allows code or commands to be executed repeatedly until the given condition is true.

```
#!/bin/bash

while [ condition ]
do
    [ command ]
done
```

To run the loop five times.

```
#!/bin/bash

declare -i var
var=0
while [ $var -lt 5 ]
do
    [ command ]
    declare var+=1
done
```

More practical usage of while are infinity loops; this loop can keep a script running until either a condition happens or if the user requests to exit (via a menu with an existing button). To create an infinite loop, we must state an always-true condition.

```
while true
do
    echo “Menu”
    echo “1) Option 1”
    echo “2) Option 2”
    echo “3) Exit”
    echo “Input something: “
    read input -n1
if [[ $input==1 ]]
then
    echo “First”
elif [[ $input==2 ]]
then
    echo “Second”
elif [[ $input==3 ]]
    exit
fi
done
```

Functions

To perform repetitive tasks more than once, use functions.

```
#!/bin/bash

function NAME()
{
    <commands>
}
```

To call the function, type its name.

```
#!/bin/bash

function NAME()
{
    echo "Cyber"
}
NAME
```

Define a function that will accept parameters while calling the function. These parameters would be represented by $1, $2, and so on, or by a $@, which stands for all given parameters.

```
#!/bin/bash

print_text () {
echo $1 $2
}
print_text Test_One Test_Two
```

In addition, return a value to the code by using the return command and then capture using the $?

The tr Command

In the vast toolkit of Linux text processing utilities, the **tr** command stands out as a powerful tool for translating or deleting characters. It's a filter that reads from standard input and writes to standard output, making it particularly useful in command pipelines. In this article, we'll delve deep into the **tr** command, exploring its functionalities, nuances, and practical applications.

Understanding the tr Command
The name **tr** stands for "translate" or "transliterate." At its core, the command is used to transform one set of characters into another. It's especially handy for tasks like converting letter cases, deleting specific characters, or squeezing repeating characters.

Basic Syntax
The basic syntax of the **tr** command is:

tr [OPTION]... SET1 [SET2]

Here, SET1 and SET2 are character sets. If only SET1 is provided, **tr** will use it to delete characters from the input. If both SET1 and SET2 are provided, **tr** will replace characters from SET1 with the corresponding characters in SET2.

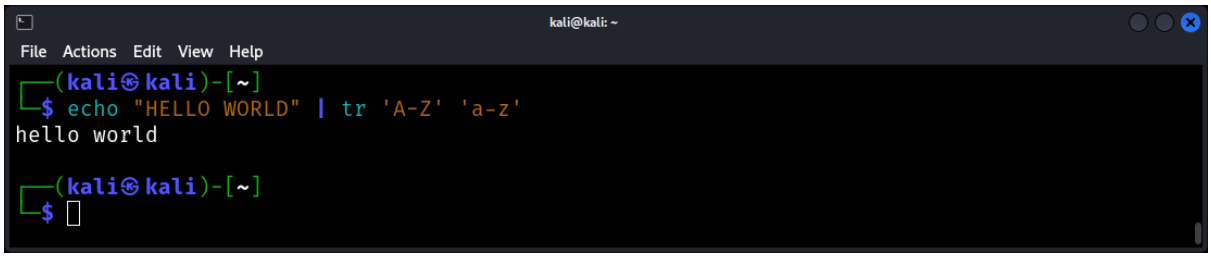
Common Options

- **-d**: Delete characters in SET1.
- **-s**: Squeeze repeating characters.
- **-c** or **-C**: Complement the set of characters in SET1.
- **-t**: Truncate SET1 to the length of SET2.

Practical Examples

1. Convert Uppercase to Lowercase
To convert all uppercase letters in a text to lowercase:

echo "HELLO WORLD" | tr 'A-Z' 'a-z'

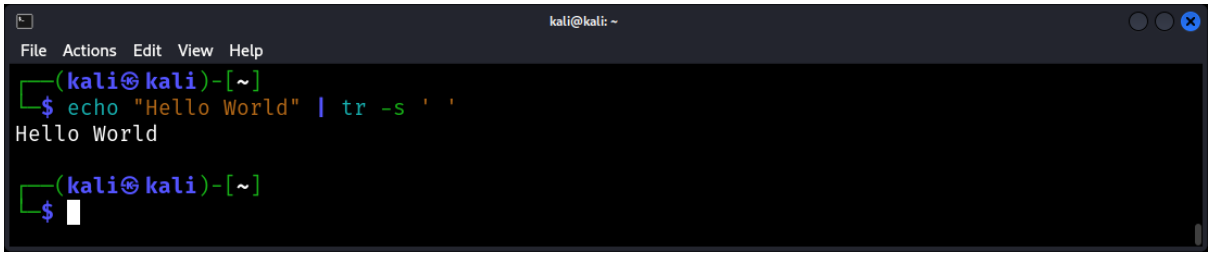


2. Delete Specific Characters
To delete all numeric characters from a text:

echo "Hello123 World456" | tr -d '0-9'

3. Squeeze Repeating Characters
To squeeze or replace repeating spaces with a single space:

echo "Hello World" | tr -s ' '



4. Complement Character Sets
To delete all characters except numeric ones:

echo "Hello123 World456" | tr -cd '0-9'

5. Translate Spaces to Tabs
To replace all spaces with tabs:

echo "Hello World" | tr ' ' '\t'

Advanced Usage: Character Classes

tr supports several character classes, which can simplify certain operations:

- **[:alnum:]**: Alphanumeric characters.
- **[:alpha:]**: Alphabetical characters.
- **[:digit:]**: Digits.
- **[:lower:]**: Lowercase letters.
- **[:upper:]**: Uppercase letters.
- **[:space:]**: Whitespace characters.

For instance, to convert all alphabetical characters to uppercase:

echo "Hello World" | tr '[:lower:]' '[:upper:]'