

1. Introduction to the operating system with key functions.

Ans: - It is a piece of software that controls computer hardware and offers a user interface through which programmes can communicate with the system. It serves as a link between the user and the hardware of the computer, aids in effective resource utilisation, and ensures a stable environment for the execution of software.

Some Key functions of Operating Systems: -

- I. **Process Management:** The OS controls how computer tasks are carried out. It distributes resources including CPU time, memory, and input/output devices to various processes to achieve equitable and effective resource utilisation.
- II. **Memory Management:** The OS controls the RAM (Random Access Memory) and virtual memory structure on the machine. To effectively handle larger programmes, it allots memory to processes, monitors memory usage, and enables memory shifting between RAM and secondary storage.
- III. **File System Management:** Data on storage devices, like as hard discs and solid-state drives, are organised and managed by the OS's file system. Users and programmes can use it to make, access, change, and remove files and folders.
- IV. **Device Management:** The operating system (OS) manages and communicates with hardware components such keyboards, scanners, printers, and network interfaces. Device drivers are made available to help with software and hardware interactions.
- V. **User Interface:** The operating system (OS) manages and communicates with hardware components such keyboards, scanners, printers, and network interfaces. Device drivers are made available to help with software and hardware interactions.
- VI. **Security Management:** Security measures are put in place by an OS to guard against malware, viruses, and unauthorised access. Through systems for authentication and authorisation, it regulates user access.
- VII. **Networking:** Many contemporary operating systems provide networking features that allow machines to communicate with one another across local networks or the internet. They control network connections and offer data-exchange methods.
- VIII. **Interrupt Handling:** The OS handles interrupts generated by hardware devices or software exceptions. Interrupts allow the OS to respond to external events and ensure proper functioning of the system.
- IX. **Scheduling:** The OS uses scheduling algorithms to determine the order in which processes are executed on the CPU. It aims to maximize CPU utilization, minimize response time, and ensure fair allocation of resources among processes.
- X. **Error Handling:** Operating systems handle errors and exceptions that occur during program execution. They provide error messages and recovery mechanisms to minimize the impact of errors on the overall system.
- XI. **Power Management:** Many modern operating systems support power-saving features to manage the energy consumption of devices like laptops and mobile devices. These features help extend battery life and reduce energy usage.
- XII. **Virtualization:** Some operating systems support virtualization technologies that allow multiple virtual machines (VMs) to run on a single physical machine. Virtualization facilitates efficient resource utilization and isolation between different computing environments.

2. Introduction to the Unix/Linux (Architecture).

Ans: - Unix and Linux are operating systems that have a similar architectural design, as Linux was inspired by the Unix operating system. The architecture of Unix/Linux is based on the principles of modularity, simplicity, and the Unix philosophy, which emphasizes small, single-purpose tools that can be combined to perform complex tasks.

The key components of the Unix/Linux architecture:

- I. **Kernel:** At the core of the Unix/Linux architecture is the kernel. It is the heart of the operating system and provides essential services to manage the computer's hardware and software resources. The kernel handles process management, memory management, device drivers, system calls, and inter-process communication.
- II. **Shell:** The shell is a command-line interface (CLI) that acts as an intermediary between the user and the kernel. Users interact with the system by entering commands into the shell, which then interprets and executes these commands. The shell allows users to run programs, manage files, and configure the system.
- III. **File System:** Unix/Linux follows a hierarchical file system structure where everything is treated as a file, including directories, devices, and even network resources. The file system provides a unified way to organize and access data on storage devices.
- IV. **Processes:** Processes are running instances of programs. The kernel manages processes by allocating resources, scheduling their execution on the CPU, and facilitating communication between them. Each process has its own memory space, ensuring isolation and security.
- V. **Memory Management:** The kernel manages the computer's memory, dividing it into different regions for kernel space and user space. It uses virtual memory techniques to map physical memory to virtual memory, allowing processes to access more memory than physically available.
- VI. **Device Drivers:** Device drivers are small software modules that allow the kernel to communicate with hardware devices. They provide an abstraction layer, enabling the kernel and applications to interact with different devices without needing to know specific hardware details.
- VII. **System Calls:** System calls are interfaces provided by the kernel to allow user-space programs to request services from the operating system. Examples include creating processes, reading/writing files, and network communication.
- VIII. **Libraries:** Unix/Linux provides libraries that contain reusable code and functions, making it easier for developers to build applications. These libraries include the C Standard Library (libc) and others that offer various functionalities.
- IX. **Networking:** Unix/Linux is renowned for its networking capabilities. It supports networking protocols and provides utilities for network configuration, communication, and security.
- X. **Shell Scripts:** Unix/Linux supports shell scripting, allowing users to create scripts that execute a series of commands. Shell scripts enhance automation and simplify complex tasks.

- XI. **User and Group Management:** Unix/Linux is a multi-user system, and the operating system manages user accounts, permissions, and access control. Each user is associated with one or more user groups.

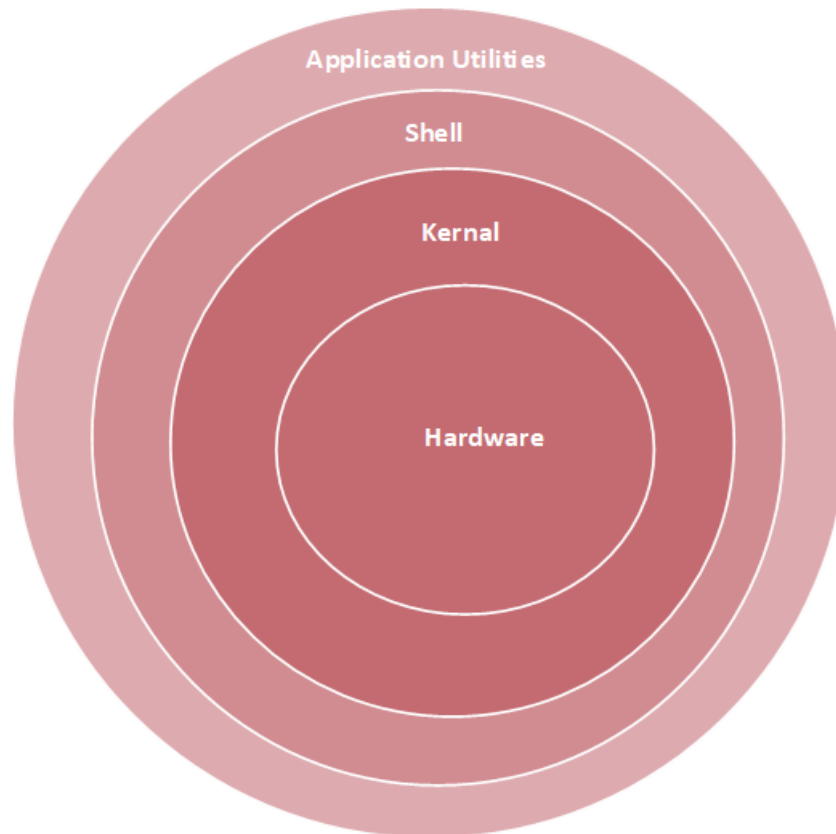


Fig: - Unix/ Linux Architecture

3. Concept of Shell.

Ans: - The concept of a shell in the context of computing refers to a command-line interface (CLI) that acts as an intermediary between the user and the operating system's kernel. It allows users to interact with the computer system by entering text-based commands. The shell interprets these commands and executes them to perform various tasks.

Key aspects of the shell concept include:

- **Command Interpretation:** The shell takes input from the user in the form of text commands and then interprets them to understand what action needs to be performed. These commands can range from simple tasks like listing files in a directory to complex operations like running scripts or managing system configurations.
- **Command Execution:** Once the shell interprets the command, it communicates with the operating system's kernel to execute the requested task. The kernel is the core part of the operating system responsible for managing hardware and software resources.

- **Flexibility:** The shell provides a highly flexible and powerful way to interact with the system. Users can chain multiple commands together, use variables, loops, and conditionals to create complex workflows, and automate tasks using shell scripts.
- **Portability:** Shells are available on various operating systems, including Unix, Linux, macOS, and Windows. Though the commands themselves may differ between shells, the concept remains consistent.
- **Environment Customization:** Users can customize their shell environment by setting environment variables, defining aliases, and configuring various settings to tailor the shell to their preferences and needs.
- **Scripting:** Shells support scripting, allowing users to create reusable scripts that contain a series of commands and can be executed as a single unit. Shell scripting is widely used for automation, system administration, and repetitive tasks.

Two popular shell implementations are:

- **Bash (Bourne Again Shell):** A widely used shell that is the default on many Unix-based systems, including Linux and macOS.
- **PowerShell:** A shell primarily used on Windows systems that provides an extensive set of features and capabilities for system administration and automation.

4. Types of Shell.

Ans: There are several types of shells available, each with its own features, syntax, and capabilities.

Some of the most used types of shells are:

- **Bash (Bourne Again Shell):** Bash is one of the most popular and widely used shells. It is the default shell for many Unix-based systems, including Linux and macOS. Bash is backward-compatible with the original Unix shell (Bourne shell) and provides an extensive set of features for scripting and interactive use.
- **Zsh (Z Shell):** Zsh is an extended version of the Bourne shell and offers additional features and customization options. It provides advanced tab completion, spelling correction, and powerful globbing capabilities. Zsh is known for its user-friendly interface and is commonly used by power users and developers.
- **Fish (Friendly Interactive Shell):** Fish is designed to be user-friendly and interactive. It has a modern and intuitive command-line interface with syntax highlighting, autosuggestions, and easy-to-remember command options. Fish aims to make the command line more accessible to new users.
- **Ksh (Korn Shell):** Korn Shell is another Unix shell that is backward-compatible with the Bourne shell. It includes many advanced features and is often used by system administrators and developers.
- **Csh (C Shell):** C Shell is a shell that provides a syntax similar to the C programming language. It was popular in the early days of Unix but has largely been replaced by more modern shells like Bash and Zsh.

- **Tcsh (TENEX C Shell):** Tcsh is an enhanced version of the C Shell, providing additional features and improvements. It offers better command-line editing capabilities and command-line history management.
- **PowerShell:** While not a traditional Unix-style shell, PowerShell is the default shell for Windows systems. It is designed to be object-oriented and includes advanced features for system administration and automation on Windows platforms.

Each shell has its own strengths and weaknesses, and users often choose the one that best suits their preferences and requirements for interactive use or scripting tasks.

5. Command structure.

Ans: The command structure in a shell refers to the format and syntax that commands must follow when entered by the user. The basic command structure typically consists of the command itself, followed by options (also called flags) and arguments.

Components of Command Structure: -

- **Command:** The actual action or program that you want to execute. It is the primary part of the command that tells the shell what operation to perform.

For example:

```
[avisek@kali]~$ ls
Desktop  Music  packages.microsoft.gpg  Templates
Documents  'my directory'  Pictures  Videos
Downloads  'my vs code directory'  Public  vscode

[avisek@kali]~$ mkdir "my vs code directory"
mkdir: cannot create directory 'my vs code directory': File exists

[avisek@kali]~$ cp
cp: missing file operand
Try 'cp --help' for more information.

[avisek@kali]~$ rmdir my vs code directory
rmdir: failed to remove 'my': No such file or directory
rmdir: failed to remove 'vs': No such file or directory
rmdir: failed to remove 'code': No such file or directory
rmdir: failed to remove 'directory': No such file or directory

[avisek@kali]~$
```

- **Options (Flags):** Optional parameters that modify the behaviour of the command. They usually start with a hyphen (-) or double hyphen (--). Options can be used to enable or disable certain features of the command.

For example:

```
ls -l # used to list all information about files and direcotry inside a file system
cp -r #used to copy files and directory inside a file system
rm -f #delete directory and its content recursively
```

- **Arguments:** Additional information required by the command to perform its action. Arguments can be filenames, directories, or any other data needed by the command to operate. Commands may accept zero, one, or multiple arguments.

For example:

```

cp file1.txt file2.txt # copy contents of file1.txt and paste it in file2.txt
mkdir my_directory #make a directory inside file system with name "my_directory"
rm file.txt # remove file file.txt

```

- **Command Line Options (Sometimes called long options):** Some commands allow for more descriptive options that start with two hyphens (--). These options are often used for more complex configurations.

For example:

```

ls --all # it will list all expended details of files and directory inside a file system
git commit --message "Initial commit" # used to commit and insert a message as "Initial Commit"

```

- **Redirection and Pipes:** In addition to the basic structure, the shell allows you to redirect input or output of commands and connect commands together using pipes (|). These mechanisms enable more complex command combinations.

For example:

```

ls > output.txt # Redirects the output of 'ls' to a file named 'output.txt'
cat file.txt | grep "keyword" # Uses a pipe to pass the output of 'cat' as input to 'grep'

```

- Command Structure can different between shells but mostly are same. To read and understand more about any command you can use “**man**” command with the command to open its manual.

For example:

```

# man coomand has been used to reade manuall of any command in details
man ls
man cp
man rm

```

6. Introduction of basic Linux commands (sudo, ls, pwd, mkdir, rmdir, rm, cd, cp, wc, mv, cmp, passwd, who, uname).

Ans: An introduction to some basic Linux commands:

- 🚦 **sudo:** Short for "superuser do," `sudo` allows authorized users to execute commands with administrative privileges (root access). It is often used for system administration tasks that require elevated permissions.

SS-1 Lab Assignment 1

```
(avisek@kali)-[~]
└─$ sudo
usage: sudo -h | -K | -k | -V
usage: sudo -v [-ABkNnS] [-g group] [-h host] [-p prompt] [-u user]
usage: sudo -l [-ABkNnS] [-g group] [-h host] [-p prompt] [-U user] [-u
user] [command [arg ...]]
usage: sudo [-ABbEHknPS] [-r role] [-t type] [-C num] [-D directory] [-g
group] [-h host] [-p prompt] [-R directory] [-T timeout] [-u
user] [VAR=value] [-i | -s] [command [arg ...]]
usage: sudo -e [-ABkNnS] [-r role] [-t type] [-C num] [-D directory] [-g
group] [-h host] [-p prompt] [-R directory] [-T timeout] [-u
user] file ...

(avisek@kali)-[~]
└─$
```

- ls: The 'ls' command lists the contents of a directory. When used without options, it displays the names of files and subdirectories in the current working directory.

```
(avisek@kali)-[~]
└─$ ls
Desktop      Music      packages.microsoft.gpg  Templates
Documents    'my directory'  Pictures                Videos
Downloads    'my vs code directory'  Public                  vscode
```

- pwd: 'pwd' stands for "print working directory." It displays the full path of the current working directory, showing the directory you are currently located in.

```
(avisek@kali)-[~]
└─$ pwd
/home/avisek
```

- mkdir: The 'mkdir' command is used to create a new directory. For example, 'mkdir my_directory' will create a directory named "my_directory."

```
(avisek@kali)-[~]
└─$ mkdir "vs directory"

(avisek@kali)-[~]
└─$
```

- rmdir: 'rmdir' is used to remove (delete) an empty directory. It only works if the directory is empty. If the directory contains any files or subdirectories, you must use the 'rm' command with appropriate options.

```
(avisek@kali)-[~]  
$ rmdir mydir  
  
(avisek@kali)-[~]  
$
```

- ✚ **rm:** The ``rm`` command is used to remove (delete) files or directories. Be careful when using this command, as it permanently deletes files, and there is no undo operation.

```
(avisek@kali)-[~]  
$ rm file1.txt  
  
(avisek@kali)-[~]  
$
```

- ✚ **cd:** The ``cd`` command is used to change the current working directory. For example, ``cd /home/ratn`` will move you to the `"/home/ratn"` directory.

```
(avisek@kali)-[~]  
$ cd  
  
(avisek@kali)-[~]  
$
```

- ✚ **cp:** The ``cp`` command is used to copy files or directories from one location to another. For example, ``cp file1.txt file2.txt`` will copy "file1.txt" to "file2.txt."

```
(avisek@kali)-[~]  
$ cp file1.txt file2.txt  
  
(avisek@kali)-[~]  
$
```

- ✚ **wc:** The ``wc`` command is used to count the number of lines, words, and characters in a file. For example, ``wc file.txt`` will display line count, word count, and character count for "file.txt."


```
(avisek@kali)-[~]  
$ wc file1.txt  
1  2 16 file1.txt  
  
(avisek@kali)-[~]  
$
```

- mv: The `mv` command is used to move or rename files or directories. To move a file, you can use `mv file.txt /path/to/destination/`. To rename a file, you can use `mv old_name.txt new_name.txt`.

```
(avisek@kali)-[~]  
$ mv file.txt download  
  
(avisek@kali)-[~]  
$
```

- cmp: The `cmp` command compares two files byte by byte and displays the first differing bytes and their byte numbers. It is useful for finding differences between two binary files.

```
(avisek@kali)-[~]  
$ cmp file1.txt file2.txt  
cmp: EOF on file2.txt which is empty  
  
(avisek@kali)-[~]  
$
```

- passwd: The `passwd` command is used to change the password for a user account. If you run it without any arguments, it will prompt you to change the current user's password.

```
(avisek@kali)-[~]  
$ passwd  
Changing password for avisek.  
Current password:
```

- who: The 'who' command displays information about currently logged-in users, including their usernames, terminal, login time, and IP address.

```
(avisek@kali)-[~]
$ who
avisek    tty7          2023-08-10 00:57 (:0)

(avisek@kali)-[~]
$
```

- uname: The 'uname' command provides information about the system's kernel and operating system. Commonly used options include 'uname -a' (displays all information), 'uname -r' (displays the kernel release), and 'uname -m' (displays the machine hardware name).

```
(avisek@kali)-[~]
$ uname
Linux

(avisek@kali)-[~]
$
```

These are just a few of the many Linux commands available. Each command may have various options and functionalities. You can access the manual by using the 'man' command followed by the command name, e.g., 'man ls', 'man cp', 'man rm', etc.

```
CP(1)                                User Commands                                CP(1)
NAME
  cp - copy files and directories

SYNOPSIS
  cp [OPTION]... [-T] SOURCE DEST
  cp [OPTION]... SOURCE... DIRECTORY
  cp [OPTION]... -t DIRECTORY SOURCE...

DESCRIPTION
  Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.
  Mandatory arguments to long options are mandatory for short options
  too.

  -a, --archive
      same as -dR --preserve=all

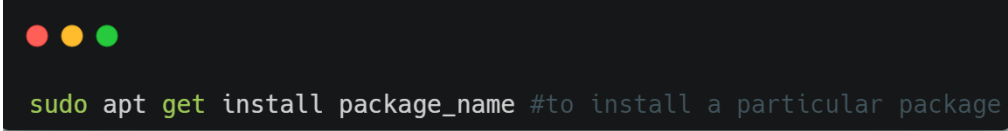
  --attributes-only
      don't copy the file data, just the attributes

  --backup[=CONTROL]
      make a backup of each existing destination file
```

7. How to install, update, upgrade and remove any package in Linux (apt-get).

Ans: In Linux, package management is typically handled through package managers like **apt-get** for Debian-based systems (e.g., Ubuntu) and **dnf** for Fedora-based systems. Here's how you can perform common package management tasks using **apt-get**:

- Installing a Package:** To install a package using **apt-get**, open a terminal, and use the following command:



```
sudo apt get install package_name #to install a particular package
```

Replace “package_name” with the name of the package you want to install. sudo is used to run the command with administrative privileges, as package installation requires root access.

- B. Updating Package Lists:** Before installing or upgrading packages, it's a good idea to update the package lists to ensure you get the latest version of packages. Use the following command:



```
sudo apt-get update # used to update packages
```

This command fetches the latest information about available packages from the repositories.

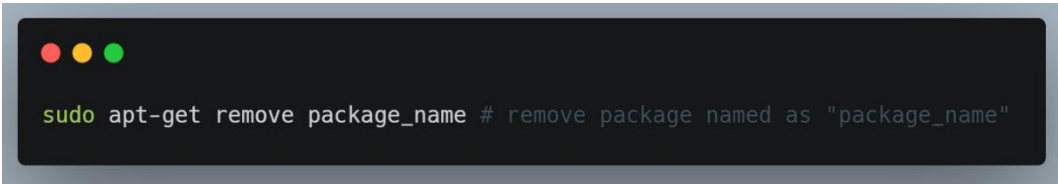
- C. Upgrading Installed Packages:** To upgrade all installed packages to their latest versions, use the following command:



```
sudo apt-get upgrade #used to upgrade repos
```

This will upgrade all installed packages that have newer versions available in the repositories.

- D. Removing a Package:** If you no longer need a package, you can remove it using the following command:



```
sudo apt-get remove package_name # remove package named as "package_name"
```

This will remove the package, but it will not remove the configuration files associated with the package.

- E. Removing a Package (Including Configuration Files):** If you want to completely remove a package along with its configuration files, use the **purge** option as follows:

```
sudo apt-get purge package_name #completly remove a package with its config files
```

- F. Cleaning Up Unneeded Packages:** Over time, your system may accumulate packages that are no longer needed. To remove these unused packages and free up disk space, use the following command:

```
sudo apt-get autoremove #it will autoremove all unnecessarily files and folders
```

8. >, >> option for directing the output of a command.

Ans: In Linux, the > and >> symbols are used for redirecting the output of a command to a file. They are known as output redirection operators.

I. > (Single Greater Than):

The > symbol is used to redirect the standard output (stdout) of a command to a file. If the file does not exist, it will be created. If the file already exists, it will be truncated (emptied), and the new output will be written to it.

Syntax:

```
command > output_file # syntax of the operation
```

Example:

```
(avisek@kali)-[~]
$ ls file2.txt
file2.txt

(avisek@kali)-[~]
$
```

In this example, the **ls** command's output will be redirected to the file **file1.txt**, overwriting any previous content in **file1.txt**.

II. >> (Double Greater Than):


The ">>" symbol is used to append the standard output of a command to a file. If the file does not exist, it will be created. If the file already exists, the new output will be appended to the end of the existing content.

Syntax:



```
command >> output_file # syntax of the operation
```

Example:



```
(avisek@kali)-[~]  
$ echo "Additional line" >> file2.txt  
  
(avisek@kali)-[~]  
$ cat file2.txt  
Additional line  
Additional line  
  
(avisek@kali)-[~]  
$
```

In this example, the **echo** command's output will be appended to the file **file1.txt**, adding a new line with the text "**Additional line**" at the end of the file.

9. Cat Command.

Ans: The **cat** command is a widely used command in Unix-like operating systems, including Linux. It is short for "concatenate" and is primarily used for displaying the contents of files or combining multiple files into a single output.

Syntax:



```
cat [OPTION]... [FILE]... /* Cat command syntax */
```

Usage:

- 1) Display the contents of a single file:



```
cat filename # to display content of filename
```

- 2) Display the contents of multiple files:



```
cat file1 file2 file # to display content of multiple files
```

This will display the contents of "file1," followed by "file2," and then "file3" in the terminal.

- 3) Concatenate multiple files into a single file:



```
cat file1 file2 > combined_file # combined file
```

This will concatenate the contents of "file1" and "file2" and save the combined content into a new file called "combined_file".

- 4) Append the contents of one file to another:



```
cat file1 >> file2 # append content of file1 to file2
```

This will append the contents of "file1" to the end of "file2."

Common Options:

- **'-n'**: Number the output lines, which adds line numbers to the display.
- **'-b'**: Number non-blank output lines, which adds line numbers to non-empty lines.
- **'-s'**: Squeeze multiple blank lines into a single blank line.

The **cat** command is simple and straightforward, but it's most used for viewing the contents of text files. For more complex operations or manipulation of file contents, other commands like **more**, **less**, **head**, or **tail** may be more suitable.

10. Compressing and archiving files (zip, tar).

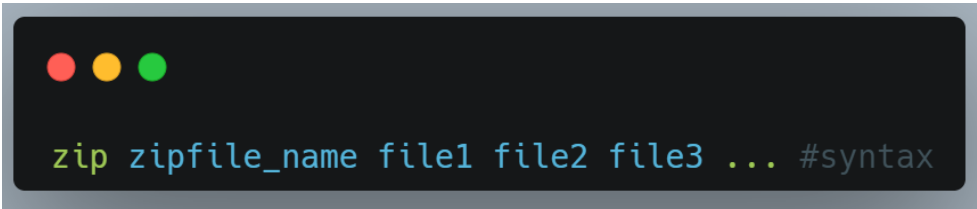
Ans: Compressing and archiving files are common tasks performed in Linux and other Unix-like operating systems. There are two main types of operations involved:

- a. **Compression:** Reducing the size of one or more files by using compression algorithms. Compressed files are often used to save storage space and make file transfer more efficient.
- b. **Archiving:** Combining multiple files into a single archive file. Archiving helps organize and bundle files together, making it easier to manage and share multiple files as a single unit.

Here are the commands to compress and archive files in Linux:

A. ZIP Compression: The **zip** command is used to compress files into a ZIP archive.

Syntax to compress files into a ZIP archive:



```
zip zipfile_name file1 file2 file3 ... #syntax
```

Example:



```
zip myarchive.zip file1.txt file2.txt #  
command
```

B. TAR Archiving: The **tar** command is used to create an archive (usually called a "tarball") of one or more files.

Syntax to create a tar archive:



```
tar -cvf tarball_name.tar file1 file2 dir1 dir2  
...
```

- **'-c'**: Create a new archive.
- **'-v'**: Verbose mode, displays the progress of archiving.
- **'-f'**: Specify the filename of the tarball.

Example:



```
tar -cvf myarchive.tar file1.txt file2.txt  
directory
```

C. GZIP Compression: The **gzip** command is used to compress a single file. It creates a new file with a **.gz** extension and removes the original file.

Syntax to compress a file with gzip:



```
gzip filename #syntax
```

Example:



```
gzip file.txt #command
```

Thank You