

## Topics To be Covered

Week	Broader Topic	Lecture	Topics	Tools to be covered
1	Process Management and Troubleshooting	51-60	<ol style="list-style-type: none"> <li>Process Management               <ol style="list-style-type: none"> <li>What is a process?</li> <li>Process Attributes</li> <li>Process Management</li> <li>Advantages</li> </ol> </li> <li>Process control and Monitoring commands</li> <li>Process control at command line</li> <li>Process management API's</li> <li>bg command               <ol style="list-style-type: none"> <li>fg command</li> <li>nice command</li> </ol> </li> <li>Troubleshooting               <ol style="list-style-type: none"> <li>ifconfig</li> <li>ping</li> <li>traceroute</li> <li>6.3.a Limitations</li> </ol> </li> <li>DNS troubleshooting tools</li> </ol>	DNS troubleshooting tools

# 1.Process Management

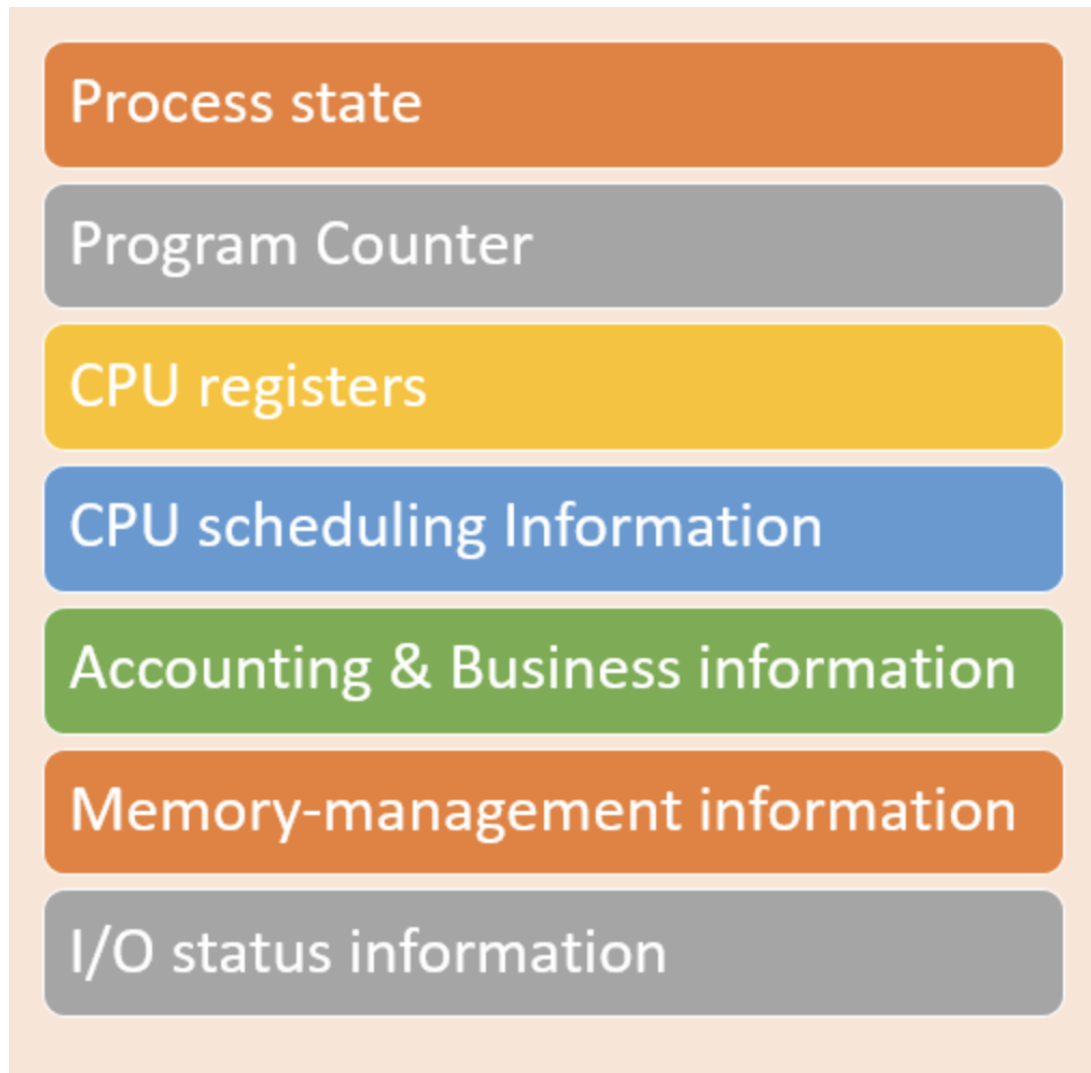
## 1.1 What is a Process?

In computing, a process refers to an instance of a running program on a computer system. It is an execution unit that represents a program or a task being carried out. A process contains the program's code, associated data, resources, and the execution context necessary for the program to run.

Here are some key characteristics of a process:

1. **Program Execution:** A process represents the execution of a program or an application. It includes the instructions of the program and any necessary data required for its execution.

2. **Process Control Block (PCB):** Each process is associated with a Process Control Block



(PCB), which contains information about the process's state, execution context, program counter, registers, memory allocation, and other process-specific details.

3. **Address Space:** A process has its own virtual address space, which consists of memory segments allocated for the program's code, data, stack, and heap. Each process has its own isolated address space, ensuring memory protection and preventing interference between processes.
4. **Resources:** Processes can utilize system resources such as CPU time, memory, files, network connections, and I/O devices. These resources are allocated to processes by the operating system and managed through process control mechanisms.

## 1.2. Process Attributes

Here are some common attributes associated with processes:

1. **Process ID (PID):** A unique identifier assigned to each process by the operating system. The PID distinguishes one process from another and is used for process management and communication.
2. **Parent Process ID (PPID):** The PID of the parent process that created the current process. It establishes the parent-child relationship between processes.
3. **User and Group IDs:** The user and group IDs associated with a process determine the permissions and access rights it has to system resources. These IDs identify the user and group that the process belongs to.
4. **CPU and Memory Usage:** Information about the amount of CPU time and memory resources consumed by a process. This includes the percentage of CPU utilization, memory usage, and virtual memory statistics.
5. **Process Priority:** A value that determines the relative importance or scheduling priority of a process. Higher priority processes are allocated more CPU time compared to lower priority processes.
6. **Execution Context:** The execution context of a process includes the program counter, register values, stack pointer, and other processor state information. It allows the process to resume execution from the point it left off.
7. **Open Files and File Descriptors:** The list of files that a process has open and the corresponding file descriptors associated with each open file. File descriptors enable the process to read from and write to files.

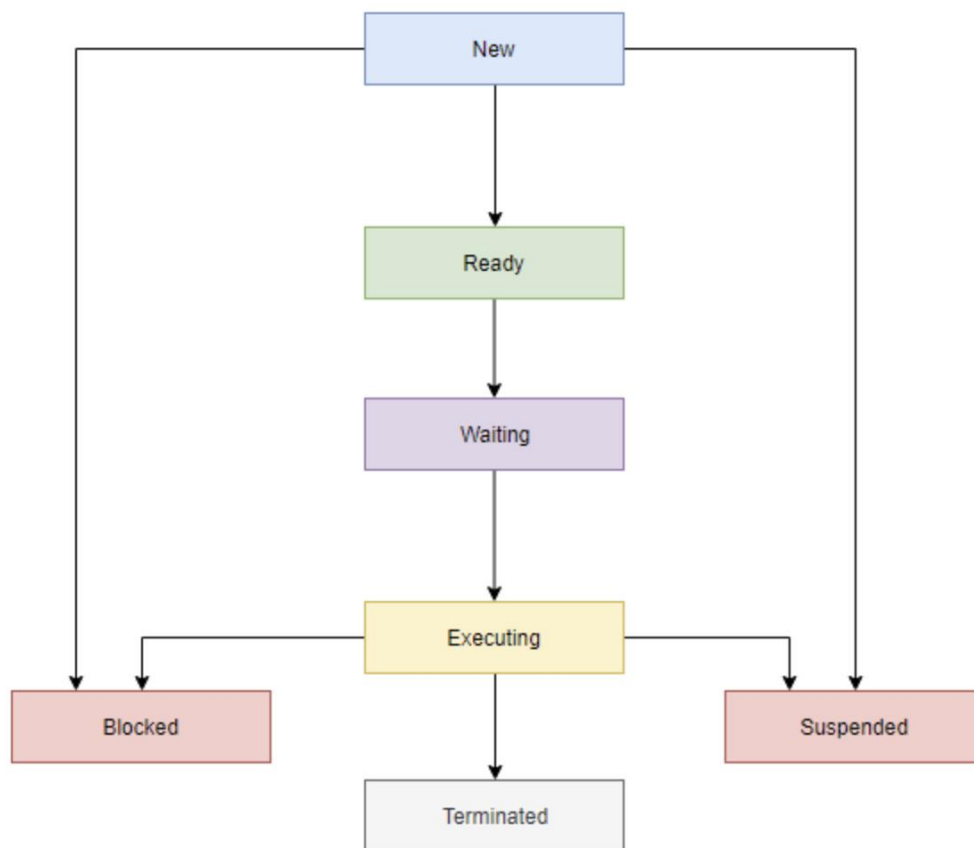
8. **Command Line Arguments:** The arguments passed to a process at the time of its creation or execution. These arguments can influence the behavior or configuration of the process.
9. **Signal Handlers:** Processes can define handlers for various signals, allowing them to respond to specific events or interruptions. Signal handlers determine the actions to be taken when a signal is received by the process.

These attributes provide valuable information about processes, their behavior, and resource utilization.

### 1.3. Process management

Refers to the activities and techniques involved in controlling and monitoring processes running on a computer system. Process management allows an operating system to manage and coordinate multiple concurrent processes efficiently. Here are some key aspects of process management:

1. **Process Creation:** Process creation involves creating new processes within the operating system. This includes allocating resources, setting up necessary data structures, and initializing the process. Processes can be created by the operating system itself or by other running processes.
2. **Process Scheduling:** Process scheduling determines which processes are allocated the CPU and for how long. The scheduling algorithm determines the order and priority in which processes are executed, ensuring fair and efficient utilization of system resources.
3. **Process States:** Processes can exist in different states during their lifecycle, such as running, waiting, ready, and terminated. The operating system manages the transitions between these states based on events and scheduling decisions.



4. **Process Control:** Process control involves managing the execution of processes, including starting, pausing, resuming, and terminating them. The operating system provides mechanisms to control and monitor processes, such as signals and system calls.
5. **Interprocess Communication (IPC):** IPC mechanisms enable processes to exchange information and synchronize their activities. Common IPC mechanisms include shared memory, pipes, sockets, message queues, and signals. They facilitate communication and coordination between processes running concurrently.
6. **Process Synchronization:** Process synchronization ensures that processes cooperate and coordinate their actions when accessing shared resources. Techniques like locks, semaphores, and monitors are used to prevent conflicts and maintain consistency.

7. **Process Termination:** Process termination involves cleaning up resources, releasing memory, and ending the execution of a process. The operating system may initiate the termination or respond to a request from the process itself or another controlling entity.
8. **Process Monitoring and Control:** The operating system provides tools and utilities to monitor and control processes. These include commands like `ps` (process status), `top`, and task managers that display information about running processes, resource usage, and allow actions like terminating or suspending processes.

Efficient process management is essential for multitasking and resource sharing in modern operating systems. It ensures optimal utilization of system resources, responsiveness, and stability by managing the execution of concurrent processes and providing mechanisms for communication and synchronization.

## 1.4 Advantages of Process Management

Process management in a computer system offers several advantages that contribute to efficient resource utilization, multitasking, and overall system stability. Here are some key advantages of process management:

1. **Resource Utilization:** Process management ensures optimal utilization of system resources, such as CPU time, memory, disk I/O, and network bandwidth. By efficiently scheduling and allocating resources among multiple processes, it maximizes the throughput and responsiveness of the system.
2. **Multitasking:** Process management enables multitasking, allowing multiple processes to run concurrently on a single system. This enables users to execute multiple programs or tasks simultaneously, improving productivity and efficiency.
3. **Process Isolation:** Each process operates in its own isolated environment, with its own address space and resources. Process isolation provides protection and prevents interference between processes, enhancing system stability and security. If one process crashes or misbehaves, it typically does not affect other processes.
4. **Process Synchronization and Communication:** Process management facilitates interprocess communication and synchronization, allowing processes to

exchange data, collaborate, and coordinate their activities. Mechanisms like shared memory, pipes, sockets, and message queues enable efficient communication and resource sharing between processes.

5. **Fault Tolerance:** Process management plays a vital role in system resilience and fault tolerance. If a process encounters an error or crashes, the operating system can handle the termination, cleanup, and recovery process. In some cases, the system may be able to restart or replace the failed process automatically.
6. **Process Prioritization:** Process management allows setting priorities for different processes, determining their relative importance in resource allocation and scheduling. Prioritization ensures that critical or high-priority processes receive sufficient resources and are executed in a timely manner.
7. **Process Control and Monitoring:** Process management provides tools and utilities for controlling, monitoring, and managing processes. System administrators can monitor the behavior, resource usage, and performance of processes, allowing for troubleshooting, optimization, and resource planning.
8. **Flexibility and Extensibility:** Process management provides a flexible framework that allows for the creation and execution of various types of processes. It supports the execution of diverse applications, services, and user programs, providing a versatile computing environment.

Overall, effective process management enhances system performance, responsiveness, and stability. It enables efficient resource utilization, supports concurrent execution of multiple tasks, and facilitates communication and synchronization between processes. These advantages collectively contribute to a robust and productive computing environment.



## 2. Process control and Monitoring commands

### 1. Process Control:

- **Process Creation:** New processes can be created using system calls such as **fork()** or **exec()**. The **fork()** system call creates a new process as a copy of the calling process, while **exec()** replaces the current process with a new program.
- **Process Termination:** Processes can be terminated by sending signals using commands like **kill** or programmatically using system calls such as **kill()** or **exit()**. The SIGKILL signal (signal number 9) is commonly used to forcefully terminate a process.

The **kill()** and **exit()** commands are related to process management and termination in the context of programming and the Linux/Unix operating systems. Here's an overview of each command:

#### 1. **kill():**

The **kill** command in Linux/Unix is used to send signals to processes, allowing you to control their behavior or request them to terminate. Here's an overview of the **kill** command:

**Syntax:** **kill** [options] <PID>

- **<PID>** represents the Process ID of the target process. You can specify the PID of the process to be signaled.
- Signals can be specified using their numeric values (e.g., 9 for SIGKILL) or their corresponding symbolic names (e.g., SIGKILL).
- By default, the kill command sends the SIGTERM signal (signal number 15) to the target process, which requests the process to terminate gracefully. If the process doesn't respond to SIGTERM, you can use more forceful signals like SIGKILL (9), which immediately terminates the process without allowing it to perform any cleanup operations.
- The kill command can also send other signals to processes for various purposes. For example, the SIGSTOP signal (19) suspends the process, and SIGCONT (18) resumes the process if it was previously suspended.
  - Options can be used to modify the behavior of the kill command. Some common options include:
    - -l: Lists the available signal names.

```
bash-3.2$ kill -l
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL
5) SIGTRAP     6) SIGABRT     7) SIGEMT      8) SIGFPE
9) SIGKILL     10) SIGBUS     11) SIGSEGV    12) SIGSYS
13) SIGPIPE    14) SIGALRM    15) SIGTERM    16) SIGURG
17) SIGSTOP    18) SIGTSTP    19) SIGCONT    20) SIGCHLD
21) SIGTTIN    22) SIGTTOU    23) SIGIO      24) SIGXCPU
25) SIGXFSZ    26) SIGVTALRM 27) SIGPROF    28) SIGWINCH
29) SIGINFO    30) SIGUSR1    31) SIGUSR2
bash-3.2$
```

- -s <signal> or --signal=<signal>: Specifies the signal to send.
- -<signal>: Sends the specified signal.
- -<signal> <PID>: Sends the specified signal to the specified process.
- -<signal> -<signal> <PID>: Sends multiple signals to the specified process.

- `-<signal> -<signal> -<signal> <PID>`: Sends multiple signals to the specified process, in a specific order.

Examples:

- To terminate a process with PID 1234: `kill 1234`
- To send the SIGKILL signal to process 5678: `kill -9 5678`
- To suspend process 9876: `kill -STOP 9876`
- To resume process 9876 if it was previously suspended: `kill -CONT 9876`

The `kill` command is a powerful tool for managing processes and controlling their behavior. It allows you to gracefully terminate processes, suspend or resume their execution, and send various other signals for process control and communication.

## 2. `exit()`:

The `exit` command is used to terminate a shell session or a script and return an exit status to the parent process. It is commonly used in shell scripts or interactive shell sessions to gracefully exit and provide a status code that indicates the success or failure of the executed script. Here's an overview of the `exit` command:

### **Syntax:** `exit [status]`

- `status` is an optional parameter that represents the exit status of the script or shell session. It is an integer value, and by convention, a status of 0 indicates successful execution, while non-zero values indicate some form of error or abnormal termination.
- When the `exit` command is encountered, the shell or script immediately terminates execution, and control is returned to the parent process.
- If the `status` parameter is omitted, the exit status is set to the status of the last executed command or the value of the `$_` variable, which stores the exit status of the most recently executed command or script.
- The exit status can be used by the calling process or script to determine the outcome of the execution and perform further actions or error handling based on the returned status.

- The `exit` command is commonly used at the end of shell scripts or in conditional statements to exit the script prematurely if certain conditions are met or to indicate specific failure scenarios.

Examples:

- Exit with a successful status (0): `exit 0`
- Exit with an error status (non-zero): `exit 1`
- Exit with the status of the last executed command: `exit $?`

```
[bash-3.2$ exit 0
exit
[nitikaraghwa@nitikas-MacBook-Air ~ %
[nitikaraghwa@nitikas-MacBook-Air ~ % exit 1

Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.

[Process completed]
```

It's important to note that the `exit` command terminates the shell or script in which it is executed, so any further statements or commands after the `exit` command will not be executed.

The `exit` command provides a way to control the flow of a script or shell session and communicate the outcome of the execution to the calling process or parent shell.

## 2. Process Monitoring and Information:

- **Process Status:** The **ps** command allows you to view information about running processes, including their PIDs, CPU usage, memory usage, and more.

The **ps** command in Linux/Unix is used to provide information about active processes running on a system. It allows you to view details such as process IDs (PIDs), resource utilization, process states, and more. Here's an overview of the **ps** command:

### Syntax: **ps** [options]

Some common options used with the **ps** command include:

- **aux:** This option displays a detailed list of all processes running on the system, including processes from all users. It shows information such as the PID, CPU and memory usage, process states, and command names.
- **ef:** This option provides a process tree view, showing the relationship between parent and child processes. It displays the processes in a hierarchical format.
- **l:** This option provides a long-format output, displaying additional details about the processes, such as the process owner, start time, terminal associated with the process, and more.
- **u:** This option displays the processes with user-oriented format, showing the username, terminal, start time, CPU and memory usage, and command.
- **--sort:** This option allows you to specify the column by which the output should be sorted. For example, **ps aux --sort=-%cpu** will sort the processes by CPU usage in descending order.
- **--forest:** This option displays the process tree in a tree-like format, highlighting the relationship between parent and child processes.
- **--pid:** This option allows you to specify one or more PIDs to filter and display information about specific processes.

Example usage:

- Display a list of all processes: **ps aux**

```

bash-3.2$ ps -ef
UID      PID     PPID    C  STIME  TTY          TIME CMD
0         1         0    0  10:56AM ??        1:57.29 /sbin/launchd
0        514         1    0  10:57AM ??        1:49.75 /usr/libexec/logd
0        516         1    0  10:57AM ??        0:00.30 /usr/libexec/UserEventAgent (System)
0        518         1    0  10:57AM ??        0:00.22 /System/Library/PrivateFrameworks/Uninstall.framework/Resources/uninstalld
0        519         1    0  10:57AM ??        1:10.11 /System/Library/Frameworks/CoreServices.framework/Versions/A/Frameworks/FSEvents.framework/Versions/A/Support/fseventsd
0        520         1    0  10:57AM ??        0:02.02 /System/Library/PrivateFrameworks/MediaRemote.framework/Support/mediaremoted
0        523         1    0  10:57AM ??        0:04.88 /usr/sbin/systemstats --daemon
0        525         1    0  10:57AM ??        0:00.00 /usr/libexec/configd
0        527         1    0  10:57AM ??        0:06.94 /System/Library/CoreServices/powerd.bundle/powerd
0        532         1    0  10:57AM ??        0:00.07 /usr/libexec/remoted
0        537         1    0  10:57AM ??        0:00.24 /usr/libexec/watchdogd
0        541         1    0  10:57AM ??        2:46.89 /System/Library/Frameworks/CoreServices.framework/Frameworks/Metadata.framework/Support/mds
0        543         1    0  10:57AM ??        0:03.54 /usr/libexec/kernelmanagerd
0        544         1    0  10:57AM ??        0:02.28 /usr/libexec/diskarbitrationd
0        548         1    0  10:57AM ??        0:30.93 /usr/sbin/syslogd
0        551         1    0  10:57AM ??        0:02.34 /usr/libexec/thermalmmonitord
0        552         1    0  10:57AM ??        0:38.67 /usr/libexec/opendirectoryd
0        553         1    0  10:57AM ??        0:02.50 /System/Library/PrivateFrameworks/ApplePushService.framework/apsd
0        554         1    0  10:57AM ??        0:19.48 /System/Library/CoreServices/launchservicesd
266       555         1    0  10:57AM ??        0:00.23 /usr/libexec/timed
213       556         1    0  10:57AM ??        0:00.15 /System/Library/PrivateFrameworks/MobileDevice.framework/Versions/A/Resources/usbmuxd -launchd
0        557         1    0  10:57AM ??        0:01.60 /usr/sbin/securityd -l
0        558         1    0  10:57AM ??        0:00.01 auditd -l
0        562         1    0  10:57AM ??        0:00.02 autofs
0        563         1    0  10:57AM ??        0:09.98 /usr/libexec/dasdd
241       566         1    0  10:57AM ??        0:01.96 /usr/sbin/distnoted daemon
0        568         1    0  10:57AM ??        0:22.99 /usr/libexec/PerfPowerServices
0        570         1    0  10:57AM ??        0:00.05 /System/Library/CoreServices/login
0        571         1    0  10:57AM ??        0:06.09 /System/Library/PrivateFrameworks/GenerationalStorage.framework/Versions/A/Support/revisond

```

- Show process tree view: `ps ef`

```

0 60365 59857 0 1:15PM ttys000 0:00.00 ps -ef
bash-3.2$
bash-3.2$
bash-3.2$
bash-3.2$ ps -p 59857
  PID TTY          TIME CMD
59857 ttys000    0:00.06 bash
bash-3.2$
bash-3.2$
bash-3.2$

```

- Display detailed information about a specific process with PID 1234: `ps -p 1234`
- Show processes owned by a specific user: `ps -u username`
- Sort processes by CPU usage: `ps aux --sort=-%cpu`
- Display process tree with highlighting: `ps --forest`

The `ps` command provides a versatile way to examine running processes and gather information about their characteristics and resource utilization. By using different options, you can customize the output to suit your specific needs.

- **Process Tree:** The `pstree` command provides a hierarchical representation of processes, showing parent-child relationships.

```
parent----child(1)----subchild (1)
      |               |--subchild (2)
      |
      |-child(2)
```

The `pstree` command in Linux/Unix is used to display a tree-like representation of running processes, showing the hierarchical relationship between parent and child processes. It provides a visual overview of the process tree structure. Here's an overview of the `pstree` command:

**Syntax:** `pstree [options]`

```

systemd—ModemManager—2*[{ModemManager}]
—NetworkManager—2*[{NetworkManager}]
—accounts-daemon—2*[{accounts-daemon}]
—acpid
—apache2—5*[apache2]
—avahi-daemon—avahi-daemon
—bluetoothd
—colord—2*[{colord}]
—cron
—cups-browsed—2*[{cups-browsed}]
—cupsd
—dbus-daemon
—fwupd—4*[{fwupd}]
—gdm3—gdm-session-wor—gdm-x-session—Xorg—{Xorg}
—gnome-session-b—3*[{gnome-+
—2*[{gdm-x-session}]
—2*[{gdm-session-wor}]
—gdm-session-wor—gdm-x-session—Xorg—{Xorg}
—gnome-session-b—ssh-agent
—2*[{gdm-x-session}]
—2*[{gdm-session-wor}]
—2*[{gdm3}]
—gnome-keyring-d—3*[{gnome-keyring-d}]
—ibus-daemon—ibus-dconf—3*[{ibus-dconf}]
—ibus-engine-sim—2*[{ibus-engine-sim}]
—ibus-extension—3*[{ibus-extension-}]
—ibus-ui-gtk3—3*[{ibus-ui-gtk3}]
—2*[{ibus-daemon}]
—ibus-x11—2*[{ibus-x11}]
—irqbalance—{irqbalance}
—2*[kerneloops]
—networkd-dispat
—nvidia-persiste
—polkitd—2*[{polkitd}]
—rsyslogd—3*[{rsyslogd}]
—rtkit-daemon—2*[{rtkit-daemon}]
—snapd—22*[{snapd}]
—switcheroo-cont—2*[{switcheroo-cont}]
—systemd—(sd-pam)
—at-spi-bus-laun—dbus-daemon
—3*[{at-spi-bus-laun}]
—at-spi2-registr—2*[{at-spi2-registr}]
—dbus-daemon

```

Common options used with the `ps` command include:



- **-p:** This option displays the PIDs (Process IDs) alongside the process names in the tree view.
- **-u:** This option includes the usernames of the process owners in the output.
- **-h:** This option highlights the currently running process.
- **-n:** This option disables vertical lines in the tree view.
- **-A:** This option uses ASCII characters for drawing the tree instead of the default Unicode characters.
- **-c:** This option compresses the output by collapsing subtrees with identical names into a single line.

Example usage:

- Display the process tree with PIDs and usernames: `pstree -p -u`
- Highlight the currently running process in the tree: `pstree -h`
- Use ASCII characters for drawing the tree: `pstree -A`
- Compress the output by collapsing subtrees: `pstree -c`

The `pstree` command provides a convenient way to visualize the hierarchical structure of processes and understand their relationships. It can be helpful for troubleshooting, identifying process dependencies, and gaining a high-level overview of the process landscape on a Linux/Unix system.

- **System Monitoring:** Tools like `top`, `htop`, offer real-time monitoring of processes, system resource usage, and performance metrics.

The **top** command in Linux/Unix is a powerful utility for monitoring and managing system processes and resource utilization in real-time. It provides a dynamic, interactive display that continuously updates to show information such as CPU usage, memory usage, running processes, and more. Here's an overview of the `top` command:

## Syntax: top

PID	COMMAND	%CPU	TIME	#TH	#Q	#PORT	MEM	PURG	CMPSR	PGRP	PPID	STATE	BOOSTS	%CPU_ME	%CPU_OTHERS	UID	FAULTS	COW	MSGSENT	MSGRECV	SYSBSD	SYSMACH	CSW
225	configd	68.1	88:22.59	13	7/2	566	7473K	08	2864K	525	1	stuck	*0(1)	19.9828	2.99941	0	20865158+	124	185843722+	55088372+	77213828+	112769612+	28817651+
517	altopd	44.4	05:14.42	9	7/1	294	15M	08	5280K	417	1	stuck	*12(1)	0.08757	0.03844	0	334861+	199	9122001+	4512792+	2773316+	1426409+	4452447+
383	WindowServer	25.8	23:34.99	28	6/1	2928	881M+	6144K+	172M	583	1	sleeping	*0(1)	0.06481	0.05480	88	5272819+	35812	12521532+	14552146+	26884487+	52527635+	18771961+
788	InternetShar	18.0	26:58.12	4	3/1	54	3873K	08	1312K	788	1	stuck	*0(1)	0.00080	17.79817	0	465324+	50818+	63586741+	31683666+	28866728+	38802795+	5417134+
9	kernel_task	15.7	18:42.81	468/8	0	0	6912K	08	8	0	0	running	*0(0)	0.00000	0.00000	0	35339	87	144871847+	139451847+	0	0	99837712+
513	mDNSResponder	14.5	21:24.03	5	3	132	6561K	08	1952K	413	1	stuck	*0(1)	0.00048	0.00000	65	933551+	144	28846296+	13348636+	36386748+	28667849+	7518793+
28587	WhatsApp Hel	11.8	08:39.17	28	1	229	468M+	08	434M+	28477	28477	sleeping	*0(1)	0.00000	0.00000	581	1133127+	516	1168549+	475432+	1577425+	2963978+	2254267+
1088	UserEventAgent	6.9	09:52.77	2	1	927	7281K	08	4168K	1888	1	sleeping	*0(1)	0.24213	0.00000	581	365137+	161	57143190+	28582587+	2314385+	35208885+	1769482+
512	runningsboard	5.7	08:09.45	7	6	408	7986K	08	1248K	612	1	sleeping	*12(1)	0.00000	4.94082	0	8839	98	645097+	988998+	16971622+	1752816+	5846148+
44683	top	4.6	00:00.88	1/1	0	33	6561K+	08	88	64683	59857	running	*0(1)	0.00000	0.00000	0	4748+	62	676783+	33836+	17277+	374369+	213+
44718	screencaptur	4.2	00:00.25	2	1	66	7738K+	752K	88	1408	1408	sleeping	*0(1)	0.16156	0.00000	581	4787+	111	2788+	974+	1545+	6168+	1728+
1397	Microsoft Ex	3.5	04:53.83	46	6	411	142M	08	122M	1397	1	sleeping	*0(1)	4.83449	0.00000	581	954188+	2103	2178074+	884844+	1986353+	1715895+	4848613+
28491	WhatsApp Hel	2.2	01:43.64	11	2	283	188M	08	38M	28477	28477	sleeping	*1(1)	0.00000	0.00000	581	52882	491	622681+	689335+	1452459+	2441372+	1495346+
575	symptoms	1.9	02:58.69	3	2	119	6321K	08	1948K	675	1	sleeping	*0(795)	0.00000	0.01385	24	17855	137	212363+	159327+	33682384+	1889228+	7879833+
3729	Activity Mon	1.5	01:14.98	7	5	525	61M	96K	41M+	8729	1	sleeping	*0(128)	0.22755	0.00000	581	74912+	866	1469377+	668687+	653287+	1947955+	446148+
5289	Google Chrom	1.4	06:06.58	45	1	923	324M+	08	178M	6289	1	sleeping	*0(698)	0.00000	0.00000	581	1432189+	4681	5383369+	2766381+	9698988+	19088188+	4331402+
576	notifyd	1.2	02:05.45	2	1	598	3457K	08	968K	576	1	sleeping	*0(1)	0.00000	0.24053	0	1992	59	1084658+	615648+	18954321+	1715223+	1484836+
18281	Terminal	1.2	00:21.68	8	2	374	166M	38M	68M	18281	1	sleeping	*0(753)	0.00756	0.00000	581	265588	1386	187082+	25544+	159938+	299382+	189381+
1285	sharingd	1.1	01:53.35	5	3	339	15M	08	8496K	1285	1	sleeping	*0(1)	0.31595	0.00000	581	638953+	273	783971+	150213+	7148299+	328452+	1745762+
3316	Google Chrom	1.1	01:29.95	16	1	146	65M+	08	26M	6289	6289	sleeping	*0(3)	0.00000	0.00000	581	141868+	1864	1851221+	568962+	3891967+	1789736+	1580861+
1181	rapport	1.0	01:30.06	4	3	216	6849K	08	248K	1181	1	sleeping	*0(1)	0.29198	0.00000	581	528553+	167	524881+	118075+	6272783+	1168811+	1897474+
1	launchd	1.0	02:06.95	5	3	2738	24M	08	6352K	1	0	sleeping	*0(0)	0.00000	0.45316	0	124847+	11128+	757988+	750911+	4885856+	2766563+	1228641+
579	AirPlayXPCHe	0.9	01:37.74	5	2	171	4353K	08	2688K	579	1	sleeping	*0(1)	0.25395	0.00000	0	519721+	140	515728+	108148+	6416803+	1143279+	1881982+
514	logd	0.8	01:58.49	4	0	1179	25M	08	15K+	514	1	sleeping	*0(1)	0.00000	0.00000	0	171782+	82	846852+	967338+	5522698+	1315219+	2439686+
515	nehalper	0.7	01:02.28	3	2	118	6745K	08	2528K	615	1	sleeping	*0(4434+)	0.00000	0.07277	0	9885	113	121361+	116453+	1491441+	242832+	313392+
44719	screencaptur	0.5	00:00.08	3	1	147	7281K+	08	64719	1	1	sleeping	*0(75+)	0.00000	0.16156	581	2088	178	748+	226+	1693+	1826+	684+
28477	WhatsApp	0.5	01:03.18	35	1	612	185M	08	159M	28477	1	sleeping	*0(345)	0.00000	0.00000	581	476733	5625	566261+	243857+	1445316+	1636266+	773911+
519	faeventd	0.4	01:23.06	12	1	153	6529K	08	2584K	519	1	sleeping	*0(1)	0.00000	0.00000	0	171553+	80	28261+	197811+	2879385+	527516+	454635+
548	syslogd	0.4	00:42.07	5	4	46	1985K	08	752K	548	1	sleeping	*0(51348+)	0.00000	0.21579	0	53718+	61	137	192865+	2475144+	465341+	632281+
18148	Google Chrom	0.4	00:40.64	18	1	556	100M	08	49M	6289	6289	sleeping	*0(5)	0.00000	0.00000	581	173445	689	411279+	254489+	498535+	737397+	477818+
2744	Google Chrom	0.4	00:10.98	17	1	283	92M+	08	50M	6289	6289	sleeping	*0(3)	0.00000	0.00000	581	126934+	3814+	65683+	37117+	93227+	121145+	99542+
1449	PandP	0.4	00:34.47	17	0	90	412M	08	489M	822	1	sleeping	*0(1)	0.00000	0.00000	74	114584	253	36744+	19347+	1254653+	32164+	592924+
5314	Google Chrom	0.3	03:43.12	25	3	328	285M	1288K	62M	6289	6289	sleeping	*1(2548)	0.00000	0.00000	581	255798	712	2432341+	2982248+	3191341+	7384846+	3467745+
589	loginwindow	0.2	00:38.55	3	1	492	32M+	08	27M	589	1	sleeping	*0(150)	0.12469	0.00000	581	285240+	367	178068+	9883	823482+	1417797+	715789+
369	mds_stores	0.2	03:38.81	6	4	187	32M	08	21M	859	1	sleeping	*0(1)	0.00000	0.26663	0	508454+	111	162253	121388+	7031583+	288788+	2121436+
222	myraid	0.2	00:24.79	39	0	90	412M	08	489M	822	1	sleeping	*0(1)	0.00000	0.00000	74	114584	253	36744+	19347+	1254653+	32164+	592924+
1886	dhcpcd	0.2	00:27.51	3	2	29	1841K	08	736K	1886	1	sleeping	*0(1)	0.05982	0.00000	-2	102478+	75	182073+	109145+	694265+	385778+	699713+
554	launchservic	0.2	00:21.22	4	3	416	8513K	08	1984K	554	1	sleeping	*0(90173)	0.00000	0.16158	0	22758	97	234149+	194159+	813188+	487535+	258447+
541	mds	0.2	02:58.38	8	5	372	43M	08	34M	541	1	sleeping	*0(1)	0.25863	0.00000	0	688728	396	232227+	163858	8561888+	518868+	1584241+
18139	Google Chrom	0.2	00:19.10	17	1	141	36M	08	22M+	6289	6289	sleeping	*0(4)	0.00000	0.00000	581	75714+	442	31840+	191784	339215+	547884+	283883+
1162	corespotligh	0.2	00:35.58	4	2	197	15M	08	12M	1162	1	sleeping	*17(1)	0.04978	0.00000	581	187942+	187	116869+	10362	1343272+	1088889+	979934+
352	opendirector	0.1	00:41.05	8	7	979	12M	128K	4688K	552	1	sleeping	*0(1)	0.04983	0.02768	0	188225+	105	286597+	214380+	538778+	955253+	
586	contextstore	0.1	00:23.08	3	2	148	6721K	08	2496K	586	1	sleeping	*0(1252)	0.00000	0.12665	0	15173	147	178967+	115455+	1146124+	852288+	489885+
527	powerd	0.1	00:07.96	3	2	128	4881K	08	1184K	527	1	sleeping	*0(1)	0.00000	0.05891	0	43463+	181	6712+	52188+	292435+	149962+	118872+
516	nsurlisession	0.1	00:43.22	3	2	95	6713K	08	4884K	616	1	sleeping	*0(399)	0.05719	0.00000	242	142266+	136	138445+	20687	1368888+	896978+	873181+

When you run the top command, it displays a live updating screen with several sections providing various system and process information. Some key sections include:

- **Summary Information:** The top section of the screen shows summary information about the system, including system uptime, load average, CPU usage, and memory usage.
- **Process List:** Below the summary information, the main section displays a list of processes currently running on the system. Each process entry includes details such as the process ID (PID), CPU usage, memory usage, process states, and more. By default, the processes are sorted by CPU usage in descending order.
- **Command Options:** The top command provides several interactive command options that you can use to modify its behavior and customize the displayed information. These options can be accessed by pressing specific keys while top is running. For example, pressing "k" allows you to send signals to selected processes, "r" renices a process, and "q" quits top.
- **Resource Summary:** At the bottom of the screen, a resource summary section displays information about CPU usage, memory usage, swap usage, and tasks. It provides an overview of the system's resource utilization.

- **Interactive Features:** `top` offers various interactive features, such as sorting the process list by different columns (e.g., CPU usage, memory usage), changing the refresh interval, filtering processes based on criteria, and managing processes (e.g., sending signals, changing process priorities).

The `top` command is highly customizable, and you can configure its behavior by using command options or modifying its configuration file (`~/.toprc`). It is a useful tool for system monitoring, performance analysis, and troubleshooting, allowing you to identify resource-intensive processes, monitor system health, and manage processes interactively.

To exit `top`, you can simply press the "q" key.

**The `htop` command** is an interactive process viewer and system monitoring tool in Linux/Unix. It provides an improved and more user-friendly interface compared to the traditional `top` command. `htop` displays system information, resource utilization, and a list of running processes in real-time. Here's an overview of the `htop` command:

Syntax: `htop`

When you run the `htop` command, it opens an interactive screen that updates dynamically to show various system and process information. Some notable features of `htop` include:

- **Colorful and Interactive Interface:** `htop` has a color-coded interface that helps distinguish different types of processes and resource usage. It allows you to navigate through the process list using keyboard shortcuts.
- **Sorting and Filtering:** `htop` provides options to sort the process list based on different columns, such as CPU usage, memory usage, and process ID. You can also apply filters to display specific processes based on search criteria.
- **Detailed Process Information:** `htop` displays detailed information about running processes, including process IDs (PIDs), CPU and memory usage, process states, execution time, and more. It also shows information about threads within each process.

- **Resource Utilization:** `htop` provides a summary of resource utilization at the top of the screen, including CPU usage, memory usage, swap usage, and load average.
- **Interactive Process Management:** You can interactively manage processes using `htop`. It allows you to send signals to selected processes, change the process priority, and even kill processes directly from the interface.
- **Customization:** `htop` offers various configuration options, including the ability to customize colors, choose columns to display, and set the update frequency.

To exit `htop`, you can simply press the "q" key.

`htop` is a popular alternative to the traditional `top` command due to its improved interface, interactivity, and additional features. It provides a comprehensive view of system performance and process activity, making it a useful tool for system monitoring, troubleshooting, and process management.

### 3. Process Control at the Command Line:

- **kill Command:** The `kill` command sends signals to processes, allowing you to terminate or control their behavior.
- **pgrep and pkill:** These commands enable searching for processes based on criteria such as process name, user, or other attributes.

The **`pgrep` command** in Linux/Unix is used to search for processes based on their names or other attributes and retrieve their corresponding Process IDs (PIDs). It is particularly useful

when you want to find the PIDs of processes matching certain criteria. Here's an overview of the `pgrep` command:

**Syntax:** `pgrep [options] <pattern>`

- `<pattern>` represents the pattern or criteria to match against process names or attributes.
  - Options can be used to modify the behavior of the `pgrep` command. Some common options include:
  - `-l` or `--list-name`: Display process names along with their PIDs.
  - `-a` or `--list-full`: Display the full command-line arguments of matching processes.
  - `-f` or `--full`: Match the pattern against the entire command line of processes instead of just the process name.
  - `-x` or `--exact`: Match the pattern against the exact process name, ensuring an exact match.
  - `-i` or `--ignore-case`: Perform a case-insensitive search.
  - `-u <username>` or `--euid <username>`: Match processes owned by the specified username.
  - `-g <group>` or `--egid <group>`: Match processes belonging to the specified group.
  - `-P <ppid>` or `--parent <ppid>`: Match processes with the specified Parent Process ID (PPID).
  - `-t <terminal>` or `--tty <terminal>`: Match processes associated with the specified terminal.

Examples:

- Find the PID of a process with a specific name: `pgrep process_name`
- Display process names and PIDs: `pgrep -l process_pattern`
- Match against the full command-line arguments: `pgrep -af argument_pattern`
- Match processes owned by a specific user: `pgrep -u username process_pattern`

- Match processes belonging to a specific group: `pgrep -g groupname process_pattern`
- Match processes with a specific Parent Process ID: `pgrep -P ppid process_pattern`

The `pgrep` command is a useful tool for identifying and working with processes based on their names or other attributes. It allows you to quickly retrieve the PIDs of processes that match specific criteria, which can be helpful for further analysis, process management, or automation tasks.

**The `pkill` command** in Linux/Unix is used to send signals to processes based on their names or other attributes, allowing you to terminate or control processes that match a specific criteria. It is essentially a more powerful variant of the `kill` command that simplifies the process of sending signals to multiple processes. Here's an overview of the `pkill` command:

**Syntax:** `pkill [options] <pattern>`

- **<pattern>** represents the pattern or criteria to match against process names or attributes.
  - Options can be used to modify the behavior of the `pkill` command. Some common options include:
  - `-f` or `--full`: Match the pattern against the entire command line of processes instead of just the process name.
  - `-x` or `--exact`: Match the pattern against the exact process name, ensuring an exact match.
  - `-i` or `--ignore-case`: Perform a case-insensitive search.
  - `-u <username>` or `--euid <username>`: Match processes owned by the specified username.
  - `-g <group>` or `--egid <group>`: Match processes belonging to the specified group.

- `-P <ppid>` or `--parent <ppid>`: Match processes with the specified Parent Process ID (PPID).
- `-t <terminal>` or `--tty <terminal>`: Match processes associated with the specified terminal.

Examples:

- Terminate processes with a specific name: `pkill process_name`
- Terminate processes with a specific command-line argument: `pkill -f argument_pattern`
- Terminate processes owned by a specific user: `pkill -u username process_pattern`
- Terminate processes belonging to a specific group: `pkill -g groupname process_pattern`
- Terminate processes with a specific Parent Process ID: `pkill -P ppid process_pattern`

By default, the `pkill` command sends the `SIGTERM` signal to the matched processes, which requests them to terminate gracefully. If necessary, you can use the `-signal` option to specify a different signal to send. For example, `pkill -HUP process_name` sends the `SIGHUP` signal to the matched processes.

It's important to exercise caution when using the `pkill` command, as terminating processes abruptly can have unintended consequences. It's recommended to test the command with appropriate options and patterns before using it in production environments or critical systems.

## 4. Process Management APIs:

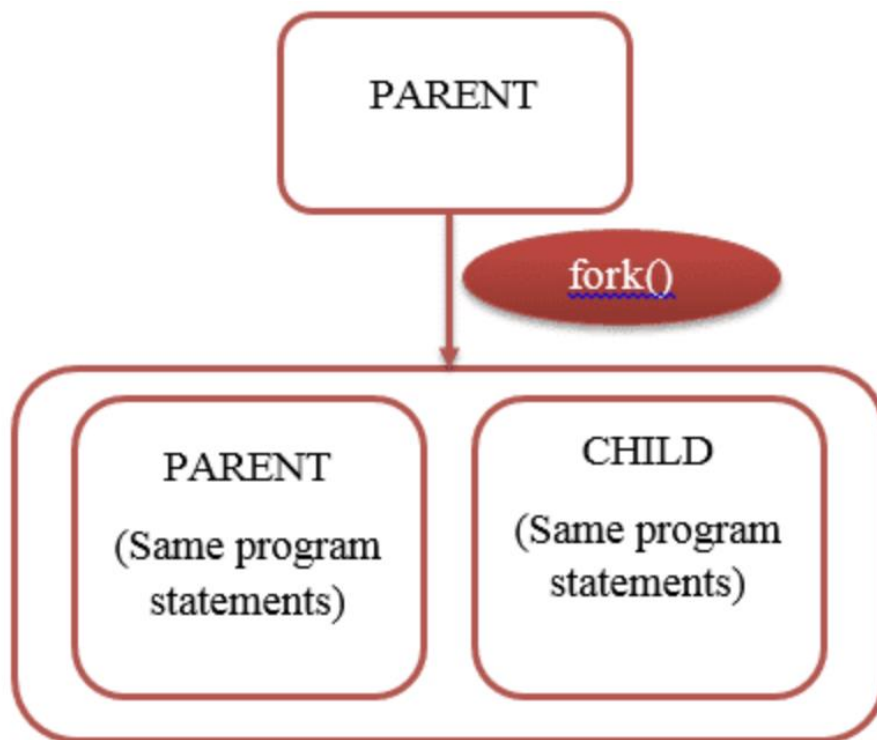
**System Call:** A system call is a mechanism provided by the operating system that allows a program or process to request services from the operating system kernel. It acts as an interface between user-level applications and the underlying operating system, providing access to low-level system functionalities and resources. System calls provide a standardized way for user processes to interact with the operating system and perform privileged operations that are typically restricted to the kernel.

- **fork():** This system call creates a new process by duplicating the existing process.

```
bash-3.2$ cat fork.c
#include <stdio.h>
#include <unistd.h>

int main()
{
    fork();
    fork();
    fork();
    fork();
    printf("Hello everyone = %d\n", getpid());
}
bash-3.2$
```





- **exec():** This system call replaces the current process with a new executable program.

```
[bash-3.2$ cat first
#include <stdio.h>
#include <unistd.h>

int main()
{
printf("PID of exc1.c program is = %d\n ",getpid());
char *args[] = {NULL};
execv("./ex2",args);
printf("Return to exc1.c");
return 0;

}
[bash-3.2$
```

```
[bash-3.2$ cat second
#include <stdio.h>
#include <unistd.h>

int main()
{
printf("We are in exc2.c program\n");
printf("PID of exc2.c is = %d\n" ,getpid());
return 0;
}
bash-3.2$ █
```

- **wait():** It allows a parent process to wait for the termination of their child process.

```
#include<unistd.h>
#include<sys/types.h>
#include<stdio.h>
#include<sys/wait.h>
int main()
{
pid_t p;
printf("before fork\n");
p=fork();
if(p==0)//child
{
printf("I am child having id %d\n",getpid());
printf("My parent's id is %d\n",getppid());
}
else//parent
{
wait(NULL);
printf("My child's id is %d\n",p);
printf("I am parent having id %d\n",getpid());
}
printf("Common\n");
}
```

Output:

```
before fork
I am child having id 8039
My parent's id is 8038
Common
My child's id is 8039
I am parent having id 8038
Common
```

- **signal():** This function allows a process to handle signals and define custom signal handlers.

## 5. bg Command

The **bg** command is used in Unix-like operating systems to move a suspended or stopped process into the background, allowing it to continue running without blocking the terminal. It is commonly used in conjunction with the **fg** (foreground) command to manage the execution of processes. Here's an overview of the **bg** command:

**Syntax:** **bg** [**job\_spec**]

- **job\_spec** represents the job specification that identifies the process you want to move to the background. It can be specified using different formats:
- **%n**: Refers to the job with the job ID **n**.
- **%string**: Refers to the job whose command line starts with **string**.

Note: If no **job\_spec** is provided, the most recent suspended or stopped job is moved to the background.

Examples:

- Move the most recent suspended or stopped job to the background: **bg**

- Move a specific job to the background using job ID: `bg %2`
- Move a specific job to the background using the command line: `bg %command_name`

When you run the `bg` command, it resumes the specified job and moves it to the background, allowing it to continue running without blocking the terminal. The process will no longer receive input from the terminal, and its output will not be displayed on the screen.

The `bg` command is often used after suspending a job using the `Ctrl+Z` keyboard shortcut or when a job has been automatically suspended due to a terminal signal. By moving a job to the background, you can free up the terminal and continue working on other tasks while the job runs in the background.

To bring a background job back to the foreground, you can use the `fg` command with the corresponding job specification (`%n` or `%string`).

## 5.1 fg command

The `fg` command is used in Unix-like operating systems to bring a background process to the foreground, making it the active process that receives input from the terminal and displays its output on the screen. It is often used in conjunction with the `bg` (background) command to manage the execution of processes. Here's an overview of the `fg` command:

**Syntax:** `fg [job_spec]`

- `job_spec` represents the job specification that identifies the process you want to bring to the foreground. It can be specified using different formats:
- `%n`: Refers to the job with the job ID `n`.
- `%string`: Refers to the job whose command line starts with `string`.

Note: If no `job_spec` is provided, the most recent suspended or stopped job is brought to the foreground.

Examples:

- Bring the most recent suspended or stopped job to the foreground: `fg`

- Bring a specific job to the foreground using job ID: `fg %2`
- Bring a specific job to the foreground using the command line: `fg %command_name`

When you run the `fg` command, it moves the specified job to the foreground, making it the active process that interacts with the terminal. The process resumes execution, and its output is displayed on the screen. You can provide input to the process using the terminal.

The `fg` command is commonly used after a process has been moved to the background using the `bg` command or when a job has been automatically suspended due to a terminal signal. By bringing a job to the foreground, you can regain control over its execution and interact with it directly.

It's important to note that a terminal can have only one active foreground process at a time. If multiple processes are running in the foreground, only the most recently brought process will receive input from the terminal, while the others will be suspended.

## 5.2. nice command

The `nice` command in Unix-like operating systems is used to adjust the priority of a process when it is launched. It allows you to modify the scheduling priority of a process, determining how much CPU time it receives relative to other processes. The `nice` command is typically used to launch processes with lower or higher priorities, depending on the desired behavior. Here's an overview of the `nice` command:

**Syntax:** `nice [options] [command [arguments]]`

- **command:** Represents the command you want to execute with adjusted priority.
- **arguments:** Optional arguments to be passed to the command.

Options:

- **-n <priority> or --adjustment=<priority>:** Specifies the priority adjustment value. A higher value indicates lower priority, and a lower value indicates higher priority. The default priority is 10.
- **-<increment>:** Increases the priority of the command by the specified increment.

- `--help`: Displays help information about the `nice` command and its options.
- `--version`: Shows the version information of the `nice` command.

Examples:

- Launch a command with a lower priority: `nice -n 10 command`
- Increase the priority of a command: `nice -5 command`
- Display the help information: `nice --help`

By default, when you execute a command without the `nice` command, it inherits the priority of its parent process. However, by using `nice`, you can explicitly adjust the priority level of a process.

Lowering the priority using a higher adjustment value (e.g., `nice -n 10 command`) means the process will have lower scheduling priority and may receive less CPU time, allowing other higher priority processes to run more frequently.

Increasing the priority using a negative adjustment value (e.g., `nice -5 command`) means the process will have a higher scheduling priority and may receive more CPU time compared to other processes with normal priorities.

Note that adjusting process priority using `nice` requires superuser privileges if you want to set a negative priority or a priority lower than the current priority of the process.

The `nice` command is useful for managing system resources and ensuring that critical processes receive sufficient CPU time while allowing less critical processes to run with lower priority.

## 55. Troubleshooting: `ifconfig` command

Troubleshooting in Linux involves identifying and resolving issues that occur within the operating system or its associated software. Here are some general steps to help you troubleshoot Linux systems:

1. **Identify the Problem:** Start by gathering information about the symptoms or errors you are experiencing. Determine if the issue is related to hardware, software, configuration, or a specific application.

2. **Check System Logs:** Review system logs, such as `/var/log/syslog`, `/var/log/messages`, or `/var/log/dmesg`, to look for error messages or warnings that could provide clues about the issue.
3. **Verify Hardware:** Ensure that all hardware components are properly connected and functioning correctly. Check for any loose connections, faulty cables, or malfunctioning devices.
4. **Verify Network Connectivity:** If the issue is related to network connectivity, check the network cables, routers, switches, and firewall settings. Use tools like `ping`, `traceroute`, or `ifconfig` to diagnose network connectivity problems.
5. **Review Configuration Files:** Examine configuration files for the affected software or system components. Check for any misconfigured settings or incorrect parameters. Common configuration files include `/etc/` directory files and application-specific configuration files located in `/etc/<application-name>/`.
6. **Test with Different Users:** Determine if the problem is specific to a particular user account. Create a new user account and test if the issue persists. This helps identify whether the problem is user-specific or system-wide.
7. **Check Disk Space:** Verify that there is enough disk space available on the system. Use the `df` command to check disk usage and ensure that critical partitions are not full.
8. **Check Memory and CPU Usage:** Use the `free` and `top` commands to check memory and CPU usage, respectively. High memory usage or CPU spikes may indicate resource-related issues.
9. **Test with Minimal Configuration:** Temporarily disable or uninstall any recently installed applications, plugins, or services that could be causing conflicts. Test the system with a minimal configuration to determine if the problem persists.
10. **Update Software:** Ensure that your system and installed software are up to date. Use package management tools like `apt`, `yum`, or `dnf` to update software packages to their latest versions.
11. **Search Online Resources and Forums:** Consult Linux documentation, forums, and community resources to see if others have experienced similar issues and if there are known solutions or workarounds available.
12. **Seek Help:** If you are unable to resolve the issue on your own, consider reaching out to online forums, Linux communities, or contacting technical support for assistance.



Remember to take caution when troubleshooting and make backups of important data before making any major changes to the system. Additionally, keep detailed notes of the steps you have taken and any changes made, as this information can be helpful if further assistance is required.

**The ifconfig command** is used in Linux and Unix-like operating systems to display or configure network interface settings. It allows you to view and modify various parameters of network interfaces on your system. Here's an overview of the ifconfig command:

**Syntax:** `ifconfig [interface] [options]`

- interface specifies the network interface to display or configure. If no interface is specified, ifconfig will display information for all active interfaces.
  - Options can be used to modify the behavior of the ifconfig command. Some common options include:
    - **up or down:** Activates or deactivates the specified interface.
    - **address <IP>:** Sets the IP address for the interface.
    - **netmask <netmask>:** Sets the netmask for the interface.
    - **broadcast <broadcast>:** Sets the broadcast address for the interface.
    - **hw <MAC>:** Sets the MAC (Media Access Control) address for the interface.
    - **mtu <MTU>:** Sets the Maximum Transmission Unit (MTU) size for the interface.
    - **promisc:** Enables promiscuous mode for the interface.
    - **multicast:** Enables multicast support for the interface.
    - **metric <value>:** Sets the routing metric for the interface.

Examples:

- Display information for all active interfaces: `ifconfig`
- Display information for a specific interface: `ifconfig eth0`
- Activate an interface: `ifconfig eth0 up`

- Deactivate an interface: `ifconfig eth0 down`
- Set an IP address: `ifconfig eth0 address 192.168.0.100`
- Set a netmask: `ifconfig eth0 netmask 255.255.255.0`
- Set a broadcast address: `ifconfig eth0 broadcast 192.168.0.255`
- Set a MAC address: `ifconfig eth0 hw ether 00:11:22:33:44:55`
- Set the MTU: `ifconfig eth0 mtu 1500`
- Enable promiscuous mode: `ifconfig eth0 promisc`
- Enable multicast support: `ifconfig eth0 multicast`
- Set the routing metric: `ifconfig eth0 metric 10`

The `ifconfig` command provides a versatile way to view and configure network interface settings. It is commonly used for troubleshooting network connectivity issues, setting IP addresses, enabling or disabling interfaces, and managing network-related parameters on Linux systems.

## 6.1 ifconfig

```
eth0      Link encap:Ethernet  HWaddr 94:de:80:87:7c:3c
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:40 Base address:0xc000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:2466 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2466 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:228069 (228.0 KB)  TX bytes:228069 (228.0 KB)

wlan0     Link encap:Ethernet  HWaddr c8:3a:35:c2:a4:cd
          inet addr:10.0.0.11  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: fe80::ca3a:35ff:fec2:a4cd/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:36408 errors:0 dropped:0 overruns:0 frame:0
          TX packets:18520 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:15698000 (15.6 MB)  TX bytes:4489507 (4.4 MB)
```

- To find the IP address of all three differently, use the command.

1. `ifconfig eth0`
2. `ifconfig lo`
3. `ifconfig wlan0`

- **Assigning IP address and Gateway**

Syntax:

`ifconfig eth0 <address> netmask <address>`

- **Enable or Disable specific interface**

1. `ifup eth0`

- **To disable specific interface**

1. `ifdown eth0`

- **Setting MTU size**

By default MTU (Maximum Transmission Unit) size is 1500, you can change size as per your wish.

1. `Ifconfig eth0 mtu xxxx`
- 2.
3. Replace xxxx with size.

## 6.2. ping command

The `ping` command is used in Linux and Unix-like operating systems to check the connectivity and reachability of a remote host on a network using the Internet Control Message Protocol (ICMP). It sends ICMP echo request packets to the specified host and waits for ICMP echo reply packets to measure the round-trip time (RTT) and determine if the host is reachable. Here's an overview of the `ping` command:

### **Syntax:** `ping [options] <host>`

- `host` specifies the hostname or IP address of the target host to ping.

Options can be used to modify the behavior of the `ping` command. Some common options include:

- `-c <count>`: Specifies the number of ICMP echo requests to send before stopping.

- `-i <interval>`: Sets the time interval (in seconds) between sending each ICMP echo request.
- `-s <size>`: Specifies the size of the ICMP echo request packet.
- `-t <ttl>`: Sets the Time-to-Live (TTL) value of the ICMP echo request packets.
- `-W <timeout>`: Sets the timeout (in seconds) for waiting for an ICMP echo reply.
- `-q`: Runs in quiet mode, displaying only a summary at the end.
- `-v`: Provides verbose output, displaying detailed information about each ICMP echo reply.
- `--help`: Displays help information about the `ping` command and its options.

Examples:

- Ping a host by its IP address: `ping 192.168.0.1`
- Ping a host by its hostname: `ping example.com`
- Limit the number of ICMP requests to 5: `ping -c 5 example.com`
- Set the interval between requests to 1 second: `ping -i 1 example.com`
- Specify the packet size as 100 bytes: `ping -s 100 example.com`
- Set the TTL value to 64: `ping -t 64 example.com`
- Display a summary at the end without verbose output: `ping -q example.com`

When you run the `ping` command, it sends ICMP echo requests to the specified host and displays the results. The output includes information about the round-trip time (RTT), packet loss, and other statistics. It is a useful tool for diagnosing network connectivity issues, checking the reachability of hosts, and troubleshooting network-related problems.

Note: In some Linux distributions, the `ping` command requires superuser (root) privileges to send ICMP echo requests. If you encounter permission errors, you can use `sudo ping` to run the command with elevated privileges.

- **ping localhost:** We can use the ping localhost name. This name will refer to our system and when we enter this command, we will say "**ping this system**".

```
bash-3.2$ ping localhost
PING localhost (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.096 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.115 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.069 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.144 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.129 ms
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.088 ms
64 bytes from 127.0.0.1: icmp_seq=6 ttl=64 time=0.098 ms
█
```

- **ping 0:** It is one of the quickest option to ping a localhost. The terminal will resolve determine the IP address and gives a response once we enter this command.

**Take a look at the ping statistics.**

Your ping responses will contain a part at the bottom titled "(host or IP) ping statistics." This part of Linux contains useful information, such as:

```
[bash-3.2$ ping 0
PING 0 (0.0.0.0): 56 data bytes
ping: sendto: Socket is not connected
ping: sendto: Socket is not connected
Request timeout for icmp_seq 0
ping: sendto: Socket is not connected
Request timeout for icmp_seq 1
ping: sendto: Socket is not connected
Request timeout for icmp_seq 2
ping: sendto: Socket is not connected
Request timeout for icmp_seq 3
ping: sendto: Socket is not connected
Request timeout for icmp_seq 4
ping: sendto: Socket is not connected
Request timeout for icmp_seq 5
ping: sendto: Socket is not connected
Request timeout for icmp_seq 6
█
```

**Transmission and reception of packets:** The number of packets the host received will be listed after "15 packets transmitted," for instance, if you ended the ping after sending 15 packets.

Packet loss is indicated by sent but unreceived packets. You'll observe a sluggish or erratic connection between your computer and the host if some packets are lost. For instance, sluggish downloads and game lag.

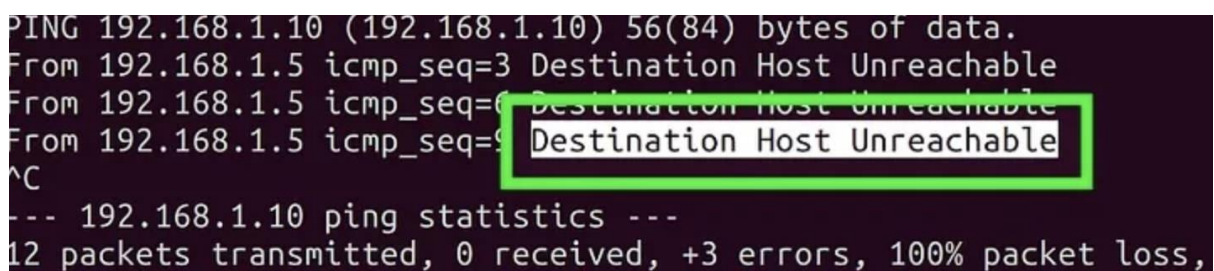
**percentage of missing packages:** For instance, you'll see "100% packet loss" in the reply if the host didn't receive any of the packets. If 2 Out of 4, you'll see "50% packet loss."

Numerous factors, such as network congestion, router issues, network hardware problems, and configuration errors on the remote server, can contribute to packet loss.[1] There might not be a problem if all packets are lost. To protect themselves from ping flooding, a sort of denial of service attack, some servers reject ICMP packets.

**RTT, or round-trip time:** This displays the time in milliseconds (MS) that it takes for each packet to be transmitted to the host and for Linux to get the response.

Multiple values for RTT are displayed: the shortest RTT, the longest RTT, the longest length of time on average, and then MDEV. Mean deviation, or RTT. A "time" value will also be displayed, which will let you know how long the entire procedure took in total.

### Interpret ping errors.



```
PING 192.168.1.10 (192.168.1.10) 56(84) bytes of data.  
From 192.168.1.5 icmp_seq=3 Destination Host Unreachable  
From 192.168.1.5 icmp_seq=4 Destination Host Unreachable  
From 192.168.1.5 icmp_seq=5 Destination Host Unreachable  
^C  
--- 192.168.1.10 ping statistics ---  
12 packets transmitted, 0 received, +3 errors, 100% packet loss,
```

You might have one of these problems when pinging a server:

- **Unidentified host:** This error message will appear if you attempt to ping a host or domain name that cannot be resolved. This could imply that the host or domain doesn't exist or that your DNS servers are unable to convert the name to an IP address.
- **Reachable host not at destination:** This error signifies that no packets could be sent to the address by your machine. This could be an indication of an issue on your network or anywhere between your network and the remote host's network on the internet.
- **The request has expired:** The issue in this instance is unquestionably unrelated to your machine or network. The ping request was sent by your machine, but there was no response. The host might not be online, it might have a network issue, or the firewall on the host might have blocked the ping.



- **Name or service not known:** The hostname or IP you are attempting to ping does not exist, as shown by this error. If the host is real, there is something wrong with your network setup.

## 6.3. traceroute command

The `traceroute` command is used in Linux and Unix-like operating systems to trace the route that packets take from your local machine to a specified destination host on a network. It provides information about the intermediate hops (routers) that the packets traverse, along with their response times. The `traceroute` command is useful for diagnosing network connectivity issues, identifying network latency, and troubleshooting network routing problems. Here's an overview of the `traceroute` command:

### Implementation of traceroute

There are multiple implementations of the `traceroute` command available for Linux and Unix-like operating systems. Here are some popular implementations:

1. **traceroute:** The original implementation of `traceroute` developed by Van Jacobson at Lawrence Berkeley National Laboratory. It is commonly found on many Linux distributions and Unix-like systems.
2. **traceroute-nanog:** An enhanced version of `traceroute` developed by the North American Network Operators' Group (NANOG). It provides additional features such as support for ICMP, TCP, and UDP protocols, and various customization options.
3. **mtr:** Short for "My traceroute," `mtr` combines the functionality of `traceroute` and `ping`. It continuously probes the network path to a destination host, displaying real-time statistics and updating the route information as it progresses. `mtr` is known for its interactive and graphical interface.
4. **tcptraceroute:** A variation of `traceroute` that uses TCP packets instead of ICMP packets. It sends TCP SYN packets to each hop along the route and uses TCP connection establishment and teardown to measure the response times. This can be useful when ICMP packets are filtered or blocked.

5. **paris-traceroute:** A variant of `traceroute` that aims to overcome the limitations of traditional `traceroute` in the presence of load balancing and firewalls. It uses techniques such as Paris-style probing to discover the actual path and measure latency even in complex network setups.

These implementations may have slight differences in features, output format, and available options. However, the basic functionality of tracing the network path and measuring response times remains consistent across all implementations.

It's important to note that the availability and default installation of these implementations can vary based on the Linux distribution or operating system version. You can typically find one or more of these implementations in the software repositories of your Linux distribution.

## Install the traceroute Command

1. `sudo apt install inetutils-traceroute`
2. `sudo apt install traceroute`

## Syntax: `traceroute [options] <host>`

- `host` specifies the hostname or IP address of the target destination.

Options can be used to modify the behavior of the `traceroute` command. Some common options include:

```
bash-3.2$ traceroute google.com
traceroute to google.com (142.251.42.110), 64 hops max, 52 byte packets
 1  reliance.reliance (192.168.29.1)  5.892 ms  2.335 ms  3.373 ms
 2  10.99.248.1 (10.99.248.1)  5.614 ms  5.801 ms  5.332 ms
 3  172.16.23.1 (172.16.23.1)  9.084 ms
    172.16.23.5 (172.16.23.5)  9.581 ms  13.889 ms
 4  192.168.112.166 (192.168.112.166)  8.933 ms
    192.168.112.168 (192.168.112.168)  9.675 ms  91.236 ms
 5  172.26.110.52 (172.26.110.52)  9.165 ms  8.685 ms  6.614 ms
 6  172.26.110.67 (172.26.110.67)  10.708 ms  11.940 ms  25.096 ms
 7  172.25.119.228 (172.25.119.228)  11.999 ms  15.929 ms
    172.25.86.126 (172.25.86.126)  8.142 ms
 8  *
```

- -n: Performs a numeric-only output, displaying IP addresses instead of hostnames.

```
bash-3.2$ traceroute -n youtube.com
traceroute to youtube.com (142.250.77.238), 64 hops max, 52 byte packets
 1  192.168.29.1  4.302 ms  3.508 ms  3.399 ms
 2  10.99.248.1  11.944 ms  5.601 ms  4.890 ms
 3  172.16.23.1  11.245 ms
    172.16.23.5  8.194 ms  8.316 ms
 4  192.168.112.168  45.381 ms  12.108 ms
    192.168.112.170  8.100 ms
 5  172.26.110.52  6.443 ms  6.677 ms  7.420 ms
 6  172.26.110.67  10.229 ms  8.501 ms  10.579 ms
 7  172.25.119.228  11.210 ms
    172.25.119.230  10.470 ms
    172.25.86.124  9.989 ms
 8  *
```

- `-q <queries>`: Specifies the number of probes per hop.
- `-w <timeout>`: Sets the timeout (in seconds) for waiting for a response from each hop.
- `-m <max_ttl>`: Sets the maximum number of hops to reach the destination.
- `-p <port>`: Specifies the destination port number.

```
[bash-3.2$ traceroute -p 20292 google.com
traceroute to google.com (142.250.207.238), 64 hops max, 52 byte packets
 1  reliance.reliance (192.168.29.1)  6.838 ms  3.509 ms  3.173 ms
 2  10.99.248.1 (10.99.248.1)  5.371 ms  4.840 ms  5.637 ms
 3  172.16.23.5 (172.16.23.5)  8.121 ms
   172.16.23.1 (172.16.23.1)  12.380 ms  7.004 ms
 4  192.168.112.168 (192.168.112.168)  7.435 ms
   192.168.112.166 (192.168.112.166)  7.761 ms
   192.168.112.168 (192.168.112.168)  8.044 ms
 5  172.26.110.52 (172.26.110.52)  8.101 ms  6.304 ms  7.370 ms
 6  172.26.110.67 (172.26.110.67)  8.434 ms  13.919 ms  40.522 ms
 7  172.25.86.126 (172.25.86.126)  6.221 ms
   172.25.119.228 (172.25.119.228)  11.878 ms
   172.25.86.126 (172.25.86.126)  9.308 ms
 8  *
```

- `-T`: Uses TCP SYN packets instead of ICMP packets for tracing the route.
- `-I`: Uses ICMP echo request packets for tracing the route.
- `-U`: Uses UDP packets for tracing the route.
- `-4`: Forces the use of IPv4 addresses only.
- `-6`: Forces the use of IPv6 addresses only.
- `--help`: Displays help information about the `traceroute` command and its options.

```
[bash-3.2$  
[bash-3.2$ traceroute --help  
Version 1.4a12+Darwin  
Usage: traceroute [-adDeFInrSvx] [-A as_server] [-f first_ttl] [-g gateway] [-i iface]  
        [-M first_ttl] [-m max_ttl] [-p port] [-P proto] [-q nqueries] [-s src_addr]  
        [-t tos] [-w waittime] [-z pausesecs] host [packetlen]  
bash-3.2$
```

#### Examples:

- Trace the route to a destination host: `traceroute example.com`
- Perform a numeric-only output: `traceroute -n example.com`
- Set the number of probes per hop to 3: `traceroute -q 3 example.com`
- Set the timeout for waiting for a response to 2 seconds: `traceroute -w 2 example.com`
- Set the maximum number of hops to 20: `traceroute -m 20 example.com`
- Use TCP SYN packets for tracing the route: `traceroute -T example.com`
- Use ICMP echo request packets for tracing the route: `traceroute -l example.com`
- Use UDP packets for tracing the route: `traceroute -U example.com`

When you run the `traceroute` command, it sends packets with gradually increasing Time-to-Live (TTL) values to the destination host. Each intermediate router decrements the TTL value and sends back an ICMP "Time Exceeded" message when the TTL reaches zero. By analyzing the sequence of routers and their response times, you can determine the path that packets take to reach the destination.

The output of the `traceroute` command typically includes the IP addresses or hostnames of the routers, along with their response times. It helps you identify network hops that may be causing latency or connectivity issues. Additionally, it can reveal if packets are taking unexpected routes or encountering network congestion.

Note: In some Linux distributions, the `traceroute` command may be replaced by `tracert` or `tracpath`. The functionality is similar, but the available options and output format may vary.

## 6.3.a Limitations of traceroute

While traceroute is a valuable tool for network troubleshooting, it does have some limitations. Here are a few limitations of traceroute:

1. **Firewall Filtering:** traceroute relies on the ICMP protocol to trace the network path. However, some firewalls and network devices may block or filter ICMP traffic, making it difficult or impossible for traceroute to obtain complete results. In such cases, traceroute may not be able to reach the target host or may provide incomplete information about the network path.
2. **Load Balancing:** In networks with load balancing mechanisms, traceroute may provide inconsistent results. Load balancing distributes traffic across multiple paths, and each traceroute probe may follow a different path. As a result, traceroute may display different intermediate hops and response times for each probe, making it challenging to determine the actual network path.
3. **Network Security Measures:** Some network security measures, such as rate limiting or rate-based traffic shaping, can affect the accuracy of traceroute. These measures can intentionally delay or drop ICMP packets, leading to inconsistent or inaccurate response times in traceroute output.
4. **Path Asymmetry:** Network paths between two hosts can be asymmetrical, meaning the path from the source to the destination can differ from the path from the destination back to the source. traceroute only shows the forward path and does not provide information about the return path. As a result, traceroute may not reflect the complete network path taken by bidirectional traffic.
5. **Timeouts and Unresponsive Hops:** Some routers or network devices may be configured to not respond to ICMP packets or have restrictive timeout settings. In such cases, traceroute probes to those hops may time out, resulting in gaps or incomplete information in the traceroute output.
6. **IPv6 Limitations:** While traceroute supports both IPv4 and IPv6, there can be limitations specific to IPv6. For example, IPv6 networks may have different routing policies or address allocation mechanisms, which can affect the accuracy of traceroute results.

It's important to keep these limitations in mind when using `traceroute` for network troubleshooting. If `traceroute` does not provide complete or expected results, it may be necessary to use alternative tools or techniques to gather additional information about the network path and diagnose connectivity issues.

## 7. DNS troubleshooting tools

When troubleshooting DNS (Domain Name System) issues, several tools are available to help diagnose and resolve problems. Here are some commonly used DNS troubleshooting tools:

1. **nslookup**: A command-line tool available on most operating systems. It allows you to query DNS servers for various types of DNS records, such as A, CNAME, MX, and NS records. `nslookup` helps you verify DNS resolution and obtain information about DNS servers and records.

The `nslookup` command is a widely used tool for troubleshooting DNS (Domain Name System) issues. It allows you to query DNS servers to obtain information about domain names, IP addresses, and other DNS records. Here's an overview of the `nslookup` command:

**Syntax:** `nslookup [options] [domain] [server]`

- **domain** (optional) specifies the domain name you want to look up. If no domain is provided, `nslookup` enters interactive mode where you can enter commands directly.
- **server** (optional) specifies the DNS server to use for the lookup. If no server is specified, `nslookup` uses the default DNS server configured on the system.

Options can be used to modify the behavior of the `nslookup` command. Some common options include:

```
[bash-3.2$ nslookup yahoo.com
Server:          2405:201:6038:1811::c0a8:1d01
Address:         2405:201:6038:1811::c0a8:1d01#53

Non-authoritative answer:
Name:   yahoo.com
Address: 74.6.231.20
Name:   yahoo.com
Address: 54.161.105.65
Name:   yahoo.com
Address: 74.6.143.26
Name:   yahoo.com
Address: 74.6.231.21
Name:   yahoo.com
Address: 98.137.11.163
Name:   yahoo.com
Address: 98.137.11.164
Name:   yahoo.com
Address: 74.6.143.25
Name:   yahoo.com
Address: 34.225.127.72

bash-3.2$
```

- `-query=<type>`: Specifies the type of DNS record to query. For example, `-query=A` for querying A records, `-query=MX` for querying MX records, etc.
- `-type=<type>`: Similar to `-query`, specifies the type of DNS record to query.
- `-class=<class>`: Specifies the class of the DNS record. The default is IN (Internet).
- `-debug`: Enables debugging mode, displaying detailed information about the DNS lookup process.
- `-timeout=<seconds>`: Sets the timeout (in seconds) for waiting for a response from the DNS server.
- `-retry=<count>`: Sets the number of retries to attempt if the DNS server does not respond.
- `-root`: Forces `nslookup` to query the root DNS servers directly.



- -all: Displays all information about the queried domain, including multiple IP addresses associated with the domain.

Examples:

1. Look up the IP address of a domain: `nslookup example.com`
2. Perform a reverse DNS lookup (IP to hostname): `nslookup 192.168.0.1`

```
bash-3.2$ nslookup 192.168.0.1
Server:                2405:201:6038:1811::c0a8:1d01
Address:               2405:201:6038:1811::c0a8:1d01#53

** server can't find 1.0.168.192.in-addr.arpa: NXDOMAIN

bash-3.2$ █
```

3. Query for a specific DNS record type: `nslookup -query=MX example.com`

```
[bash-3.2$ nslookup -query=mx www.yahoo.com
Server:                2405:201:6038:1811::c0a8:1d01
Address:               2405:201:6038:1811::c0a8:1d01#53

Non-authoritative answer:
www.yahoo.com    canonical name = new-fp-shed.wg1.b.yahoo.com.

Authoritative answers can be found from:
wg1.b.yahoo.com
    origin = yf1.a1.b.yahoo.net
    mail addr = hostmaster.yahoo-inc.com
    serial = 1687346650
    refresh = 30
    retry = 30
    expire = 86400
    minimum = 300

bash-3.2$ █
```

#### 4. Specify a custom DNS server: `nslookup example.com 8.8.8.8`

- Enter interactive mode: `nslookup`
- In interactive mode, you can enter domain names or IP addresses and receive immediate results without specifying the `nslookup` command each time.

When you run the `nslookup` command, it sends a DNS query to the specified DNS server and displays the results. The output includes information about the queried domain, such as its IP address, associated DNS records, and the DNS server that provided the response.

`nslookup` is a helpful tool for verifying DNS resolution, troubleshooting DNS configurations, and obtaining DNS-related information.

Note: The `nslookup` command is being phased out in favor of `dig`, which provides more advanced functionality and flexibility for DNS lookups.

7. **dig:** Short for "domain information groper," `dig` is a versatile command-line tool that provides detailed DNS information. It allows you to query DNS servers for specific

record types, set custom DNS server, and perform advanced DNS lookups. `dig` is widely used for troubleshooting and gathering detailed DNS information.

8.

The `dig` command is a powerful and versatile tool for querying DNS (Domain Name System) servers. It provides detailed information about DNS records, performs advanced DNS lookups, and aids in DNS troubleshooting. Here's an overview of the `dig` command:

```
[bash-3.2$ dig google.com

; <<>> DiG 9.10.6 <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 45847
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1280
;; QUESTION SECTION:
;google.com.                IN      A

;; ANSWER SECTION:
google.com.                139     IN      A      142.250.207.238

;; Query time: 12 msec
;; SERVER: 2405:201:6038:1811::c0a8:1d01#53(2405:201:6038:1811::c0a8:1d01)
;; WHEN: Wed Jun 21 17:14:20 IST 2023
;; MSG SIZE rcvd: 55

bash-3.2$
```

- The first column lists the **name of the server** that was queried
- The second column is the **Time to Live**, a set timeframe after which the record is refreshed
- The third column shows the **class of query** – in this case, “IN” stands for Internet
- The fourth column displays the **type of query** – in this case, “A” stands for an A (address) record
- The final column displays the **IP address** associated with the domain name

**Syntax:** `dig [options] [domain] [type] [server]`

- **domain** (optional) specifies the domain name you want to look up. If no domain is provided, `dig` enters interactive mode where you can enter commands directly.
- **type** (optional) specifies the type of DNS record to query. If not specified, `dig` defaults to an A record lookup.
- **server** (optional) specifies the DNS server to use for the lookup. If no server is specified, `dig` uses the default DNS server configured on the system.

Options can be used to modify the behavior of the `dig` command. Some common options include:

- **+short:** Provides a concise output, displaying only the queried information without additional details.
- **+trace:** Performs a trace route-like operation, showing the DNS resolution path from root servers to the queried domain.
- **+query=<type>:** Specifies the type of DNS record to query. For example, **+query=A** for querying A records, **+query=MX** for querying MX records, etc.
- **+class=<class>:** Specifies the class of the DNS record. The default is **IN** (Internet).
- **+timeout=<seconds>:** Sets the timeout (in seconds) for waiting for a response from the DNS server.
- **+retry=<count>:** Sets the number of retries to attempt if the DNS server does not respond.
- **@<server>:** Specifies a custom DNS server to use for the lookup.

Examples:

1. Look up the IP address of a domain: `dig example.com`
2. Perform a reverse DNS lookup (IP to hostname): `dig -x 192.168.0.1`
3. Query for a specific DNS record type: `dig example.com MX`
4. Specify a custom DNS server: `dig example.com @8.8.8.8`

```
[bash-3.2$ dig @8.8.8.8 google.com

; <<>> DiG 9.10.6 <<>> @8.8.8.8 google.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 23939
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags;; udp: 512
;; QUESTION SECTION:
;google.com.                IN      A

;; ANSWER SECTION:
google.com.                29      IN      A      142.250.192.206

;; Query time: 19 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Wed Jun 21 17:15:38 IST 2023
;; MSG SIZE  rcvd: 55
```

##### 5. Perform a trace route-like operation: `dig +trace example.com`

When you run the `dig` command, it sends a DNS query to the specified DNS server and displays the results. The output provides detailed information about the queried domain, including the IP address, associated DNS records, authoritative DNS servers, TTL (time-to-live) values, and more. `dig` is widely used for advanced DNS troubleshooting, gathering comprehensive DNS information, and verifying DNS configurations.

`dig` offers extensive capabilities beyond basic DNS lookups and is commonly preferred over other DNS query tools such as `nslookup`. It provides more control over query types, DNS server selection, and output customization.

9. **host**: Another command-line tool that performs DNS lookups and provides DNS-related information. It can retrieve A, AAAA, CNAME, MX, NS, and other DNS record types. **host** is often used for quick DNS lookups and basic troubleshooting.

The **host** command is a command-line tool used for DNS (Domain Name System) lookups in Linux and Unix-like operating systems. It is primarily used to query DNS servers for information about domain names and IP addresses. Here's an overview of the **host** command:

```
bash-3.2$ host google.com
google.com has address 142.250.207.238
google.com has IPv6 address 2404:6800:4009:832::200e
google.com mail is handled by 10 smtp.google.com.
bash-3.2$
```

**Syntax:** `host [options] [domain] [server]`

- **domain** (optional) specifies the domain name you want to look up. If no domain is provided, **host** enters interactive mode where you can enter commands directly.
- **server** (optional) specifies the DNS server to use for the lookup. If no server is specified, **host** uses the default DNS server configured on the system.

Options can be used to modify the behavior of the **host** command. Some common options include:

- **-t <type>**: Specifies the type of DNS record to query. For example, **-t A** for querying A records, **-t MX** for querying MX records, etc.
- **-v**: Enables verbose output, providing more detailed information about the lookup process.

- -a: Performs a reverse DNS lookup (IP to hostname).

```
bash-3.2$ host -a google.com
Trying "google.com"
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 22455
;; flags: qr rd ra; QUERY: 1, ANSWER: 8, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;google.com.                IN      ANY

;; ANSWER SECTION:
google.com.                273     IN      A       142.250.207.238
google.com.                262     IN      AAAA    2404:6800:4002:82f::200e
google.com.                198     IN      MX      10 smtp.google.com.
google.com.                18      IN      SOA      ns1.google.com. dns-admin.google.com. 541857717 900 900 1800 60
google.com.                32229   IN      NS       ns3.google.com.
google.com.                32229   IN      NS       ns1.google.com.
google.com.                32229   IN      NS       ns2.google.com.
google.com.                32229   IN      NS       ns4.google.com.

Received 211 bytes from 2405:201:6038:1811::c0a8:1d01#53 in 44 ms
bash-3.2$
```

- -4 or -6: Forces the use of IPv4 or IPv6, respectively.
- -W <seconds>: Sets the timeout (in seconds) for waiting for a response from the DNS server.
- -C: Disables DNS caching and performs a fresh lookup.

Examples:

1. Look up the IP address of a domain: `host example.com`

6. Perform a reverse DNS lookup (IP to hostname): `host 192.168.0.1`

```
bash-3.2$ host 192.168.0.1
Host 1.0.168.192.in-addr.arpa. not found: 3(NXDOMAIN)
bash-3.2$
```

7. Query for a specific DNS record type: `host -t MX example.com`
8. Specify a custom DNS server: `host example.com 8.8.8.8`

When you run the `host` command, it sends a DNS query to the specified DNS server and displays the results. The output typically includes the queried domain's IP address, associated DNS records, and the DNS server that provided the response. `host` is a useful tool for verifying DNS resolution, obtaining DNS-related information, and troubleshooting DNS issues.

Note that the `host` command may have slight variations in functionality and output format depending on the operating system and its version. It is one of the commonly available DNS lookup tools alongside `nslookup` and `dig`.

10. **ping:** Although primarily used for network connectivity testing, `ping` can also help identify DNS-related issues. By pinging a domain name, `ping` attempts to resolve the domain's IP address. If the ping fails or resolves to an unexpected IP address, it may indicate a DNS problem.
11. **traceroute:** As discussed earlier, `traceroute` can assist in troubleshooting DNS by revealing the network path to a destination host. It can help identify any routing issues or misconfigurations that may be affecting DNS resolution.
12. **tcpdump:** A packet capture tool that allows you to capture and analyze network traffic. By capturing DNS traffic, you can examine DNS queries, responses, and identify any issues or anomalies. `tcpdump` provides deep visibility into DNS traffic for advanced troubleshooting.



The `tcpdump` command is a powerful network packet capture and analysis tool available in Unix-like operating systems. It allows you to capture and display network traffic in real-time or analyze previously captured packet capture files. `tcpdump` is commonly used for network troubleshooting, protocol analysis, security analysis, and network monitoring.

Here's an overview of the `tcpdump` command:

**Syntax:** `tcpdump [options] [expression]`

```

PTR (QU)? _pd1-datastream_tcp.local. PTR (QU)? _uscasts_tcp.local. PTR (QU)? _ippusb_tcp.local. PTR (QU)? _printer_tcp.local. PTR (QU)? _scanner_tcp.local. PTR (QU)? _ptp_tcp.local. PTR (QU)? _ipp
_tcp.local. PTR (QU)? _ipp_tcp.local. PTR (QU)? _googlecast_tcp.local. PTR (QU)? _apple-mobdev_tcp.local. PTR (QU)? 93a207db_sub_apple-mobdev2_tcp.local. PTR (QU)? _apple-pairable_tcp.local. PTR
U)? _companion-link_tcp.local. PTR (QU)? _afpovertcp_tcp.local. PTR (QU)? _smb_tcp.local. PTR (QU)? _adisk_tcp.local. (382)
17:28:38.670794 IP6 nitikas-macbook-air.local.mdns > ff02::fb.mdns: 0 [20q] [1au] PTR (QU)? lb_dns-sd_udp.local. PTR (QU)? _airport_tcp.local. PTR (QU)? _rdlink_tcp.local. PTR (QU)? _uscan_tcp.local.
PTR (QU)? _pd1-datastream_tcp.local. PTR (QU)? _uscasts_tcp.local. PTR (QU)? _ippusb_tcp.local. PTR (QU)? _printer_tcp.local. PTR (QU)? _scanner_tcp.local. PTR (QU)? _ptp_tcp.local. PTR (QU)? _ipp
_tcp.local. PTR (QU)? _ipp_tcp.local. PTR (QU)? _googlecast_tcp.local. PTR (QU)? _apple-mobdev_tcp.local. PTR (QU)? 93a207db_sub_apple-mobdev2_tcp.local. PTR (QU)? _apple-pairable_tcp.local. PTR
U)? _companion-link_tcp.local. PTR (QU)? _afpovertcp_tcp.local. PTR (QU)? _smb_tcp.local. PTR (QU)? _adisk_tcp.local. (388)
17:28:38.670908 IP6 nitikas-macbook-air.local.mdns > ff02::fb.mdns: 0 [20q] [1au] PTR (QU)? lb_dns-sd_udp.local. PTR (QU)? _airport_tcp.local. PTR (QU)? _rdlink_tcp.local. PTR (QU)? _uscan_tcp.local.
PTR (QU)? _pd1-datastream_tcp.local. PTR (QU)? _uscasts_tcp.local. PTR (QU)? _ippusb_tcp.local. PTR (QU)? _printer_tcp.local. PTR (QU)? _scanner_tcp.local. PTR (QU)? _ptp_tcp.local. PTR (QU)? _ipp
_tcp.local. PTR (QU)? _ipp_tcp.local. PTR (QU)? _googlecast_tcp.local. PTR (QU)? _apple-mobdev_tcp.local. PTR (QU)? 93a207db_sub_apple-mobdev2_tcp.local. PTR (QU)? _apple-pairable_tcp.local. PTR
U)? _companion-link_tcp.local. PTR (QU)? _afpovertcp_tcp.local. PTR (QU)? _smb_tcp.local. PTR (QU)? _adisk_tcp.local. (388)
17:28:38.670964 IP 192.168.29.56.mdns > 224.0.0.251.mdns: 0 [1a] [20q] [1au] PTR (QU)? lb_dns-sd_udp.local. PTR (QU)? _airport_tcp.local. PTR (QU)? _rdlink_tcp.local. PTR (QU)? _uscan_tcp.local. PT
R (QU)? _pd1-datastream_tcp.local. PTR (QU)? _uscasts_tcp.local. PTR (QU)? _ippusb_tcp.local. PTR (QU)? _printer_tcp.local. PTR (QU)? _scanner_tcp.local. PTR (QU)? _ptp_tcp.local. PTR (QU)? _ipp
_tcp.local. PTR (QU)? _ipp_tcp.local. PTR (QU)? _googlecast_tcp.local. PTR (QU)? _apple-mobdev_tcp.local. PTR (QU)? 93a207db_sub_apple-mobdev2_tcp.local. PTR (QU)? _apple-pairable_tcp.local. PTR
U)? _companion-link_tcp.local. PTR (QU)? _afpovertcp_tcp.local. PTR (QU)? _smb_tcp.local. PTR (QU)? _adisk_tcp.local. (413)
17:28:38.670999 IP6 nitikas-macbook-air.local.mdns > ff02::fb.mdns: 0 [1a] [20q] [1au] PTR (QU)? lb_dns-sd_udp.local. PTR (QU)? _airport_tcp.local. PTR (QU)? _rdlink_tcp.local. PTR (QU)? _uscan_tcp
ocal. PTR (QU)? _pd1-datastream_tcp.local. PTR (QU)? _uscasts_tcp.local. PTR (QU)? _ippusb_tcp.local. PTR (QU)? _printer_tcp.local. PTR (QU)? _scanner_tcp.local. PTR (QU)? _ptp_tcp.local. PTR (QU)? _ipp
pps_tcp.local. PTR (QU)? _ipp_tcp.local. PTR (QU)? _googlecast_tcp.local. PTR (QU)? _apple-mobdev_tcp.local. PTR (QU)? 93a207db_sub_apple-mobdev2_tcp.local. PTR (QU)? _apple-pairable_tcp.local.
PTR (QU)? _companion-link_tcp.local. PTR (QU)? _afpovertcp_tcp.local. PTR (QU)? _smb_tcp.local. PTR (QU)? _adisk_tcp.local. (413)
17:28:38.671138 IP6 nitikas-macbook-air.local.mdns > ff02::fb.mdns: 0 [20q] [1au] PTR (QU)? lb_dns-sd_udp.local. PTR (QU)? _airport_tcp.local. PTR (QU)? _rdlink_tcp.local. PTR (QU)? _uscan_tcp.local.
PTR (QU)? _pd1-datastream_tcp.local. PTR (QU)? _uscasts_tcp.local. PTR (QU)? _ippusb_tcp.local. PTR (QU)? _printer_tcp.local. PTR (QU)? _scanner_tcp.local. PTR (QU)? _ptp_tcp.local. PTR (QU)? _ipp
_tcp.local. PTR (QU)? _ipp_tcp.local. PTR (QU)? _googlecast_tcp.local. PTR (QU)? _apple-mobdev_tcp.local. PTR (QU)? 93a207db_sub_apple-mobdev2_tcp.local. PTR (QU)? _apple-pairable_tcp.local. PTR
U)? _companion-link_tcp.local. PTR (QU)? _afpovertcp_tcp.local. PTR (QU)? _smb_tcp.local. PTR (QU)? _adisk_tcp.local. (388)
17:28:38.671201 IP6 nitikas-macbook-air.local.mdns > ff02::fb.mdns: 0 [20q] [1au] PTR (QU)? lb_dns-sd_udp.local. PTR (QU)? _airport_tcp.local. PTR (QU)? _rdlink_tcp.local. PTR (QU)? _uscan_tcp.local.
PTR (QU)? _pd1-datastream_tcp.local. PTR (QU)? _uscasts_tcp.local. PTR (QU)? _ippusb_tcp.local. PTR (QU)? _printer_tcp.local. PTR (QU)? _scanner_tcp.local. PTR (QU)? _ptp_tcp.local. PTR (QU)? _ipp
_tcp.local. PTR (QU)? _ipp_tcp.local. PTR (QU)? _googlecast_tcp.local. PTR (QU)? _apple-mobdev_tcp.local. PTR (QU)? 93a207db_sub_apple-mobdev2_tcp.local. PTR (QU)? _apple-pairable_tcp.local. PTR
U)? _companion-link_tcp.local. PTR (QU)? _afpovertcp_tcp.local. PTR (QU)? _smb_tcp.local. PTR (QU)? _adisk_tcp.local. (388)
17:28:38.671313 IP6 nitikas-macbook-air.local.mdns > ff02::fb.mdns: 0 [20q] [1au] [0/1] TXT "LKDC:SHA1.905288C18507C9F427F57268AF99EBB15E74BCCD" (133)
17:28:38.671351 IP6 nitikas-macbook-air.local.mdns > ff02::fb.mdns: 0 [20q] [1au] [0/1] TXT "LKDC:SHA1.905288C18507C9F427F57268AF99EBB15E74BCCD" (139)
17:28:38.671413 IP6 nitikas-macbook-air.local.mdns > ff02::fb.mdns: 0 [20q] [1au] [0/1] TXT "LKDC:SHA1.905288C18507C9F427F57268AF99EBB15E74BCCD" (139)

```

- **options:** Various options can be used to control the behavior of `tcpdump`. Some common options include:
  - **-i <interface>:** Specifies the network interface to capture packets from. For example, `-i eth0` captures packets from the `eth0` interface.
  - **-c <count>:** Sets the number of packets to capture before exiting.
  - **-w <file>:** Writes the captured packets to a file for later analysis.
  - **-r <file>:** Reads packets from a previously captured packet capture file.
  - **-n:** Displays IP addresses and port numbers instead of resolving them to hostnames and service names.
  - **-X:** Displays packet contents in both hexadecimal and ASCII formats.

- `-s <snaplen>`: Sets the snapshot length, determining the amount of packet data to capture.
- `-v`: Enables verbose output, providing more detailed information about the captured packets.
- `expression` (optional): Specifies a filter expression to selectively capture packets based on specific criteria, such as source/destination IP address, port numbers, protocol types, packet size, etc. The expression syntax follows the Berkeley Packet Filter (BPF) syntax.

Examples:

1. Capture packets on a specific network interface: `tcpdump -i eth0`
13. Capture and display a limited number of packets: `tcpdump -c 10`
14. Capture packets and write them to a file: `tcpdump -w capture.pcap`
15. Read and analyze a previously captured packet capture file: `tcpdump -r capture.pcap`
16. Capture packets from a specific source or destination IP address: `tcpdump host 192.168.0.1`
17. Capture packets for a specific port: `tcpdump port 80`
18. Capture packets for a specific protocol: `tcpdump icmp`

When you run the `tcpdump` command, it starts capturing packets based on the specified options and expression. The captured packets are displayed in real-time or analyzed from a captured packet capture file. The output includes information such as source and destination IP addresses, port numbers, protocol types, packet payloads, and more.

`tcpdump` provides a flexible and comprehensive way to capture and analyze network traffic for various purposes.

It's important to note that `tcpdump` requires root or sudo privileges to capture packets on most systems, as it operates at a low level of the networking stack.

13. **Wireshark:** A powerful network protocol analyzer that can capture and analyze network traffic, including DNS packets. Wireshark provides a graphical interface for analyzing DNS traffic in real-time or from captured packet captures. It helps you inspect DNS queries, responses, and diagnose complex DNS issues.

These tools can provide valuable insights into DNS resolution problems and assist in troubleshooting DNS-related issues. Depending on the specific problem, one or more of these tools may be useful in diagnosing and resolving DNS issues.

Install Wirehark:

**sudo add-apt-repository ppa:wireshark-dev/stable**

The Wireshark command is used to launch the Wireshark network protocol analyzer application from the command line. Wireshark is a popular open-source tool for capturing, analyzing, and inspecting network packets. It provides a graphical user interface (GUI) for capturing and analyzing network traffic in real-time or from captured packet capture files. Here's an overview of the Wireshark command:

**Syntax: wireshark [options] [file]**

- **options:** Various options can be used to control the behavior of Wireshark. Some common options include:
  - **-i <interface>:** Specifies the network interface to capture packets from. For example, **-i eth0** captures packets from the eth0 interface.
  - **-r <file>:** Opens a previously captured packet capture file for analysis.
  - **-k:** Starts packet capturing immediately upon opening Wireshark.
  - **-n:** Disables name resolution and displays IP addresses and port numbers instead of hostnames and service names.
  - **-f <capture filter>:** Applies a capture filter to capture only specific packets based on defined criteria.
  - **-Y <display filter>:** Applies a display filter to filter and display specific packets based on defined criteria.

- **file** (optional): Specifies a packet capture file to open and analyze in Wireshark. This can be a file previously captured using Wireshark or other packet capture tools.

Examples:

1. Launch Wireshark with the GUI: `wireshark`
19. Capture packets on a specific network interface: `wireshark -i eth0`
20. Open and analyze a previously captured packet capture file: `wireshark -r capture.pcap`
21. Start capturing packets immediately upon opening Wireshark: `wireshark -k`
22. Capture and display packets matching a specific filter: `wireshark -f "port 80"`
23. Apply a display filter to analyze specific packets: `wireshark -Y "ip.src == 192.168.0.1"`

When you run the Wireshark command with the appropriate options and file, the Wireshark application is launched, providing a graphical user interface for capturing and analyzing network packets. You can interact with the captured packets, apply filters, perform detailed analysis, view packet details, and perform various network protocol troubleshooting tasks.

Please note that the availability and functionality of the Wireshark command may vary depending on the operating system and the way Wireshark is installed. Additionally, running Wireshark typically requires root or sudo privileges to capture packets on most systems due to its low-level interaction with the network stack.