CompTIA Linux+ XK0-006
Study Guide

Introduction

- **Introduction**
  - Linux+ Overview
    - Intermediate-level certification
    - Part of CompTIA IT Support Specialist and Network Specialist career paths
    - Focuses on configuring, managing, operating, and troubleshooting Linux server environments
    - Prepares for roles including Network Administrator, Linux Engineer, Solution Architect, Web Administrator, Data Architect, Cybersecurity Engineer, Data Scientist, and Penetration Tester
  - Experience Assumptions
    - Assumes 12 months of hands-on experience with Linux servers
    - Assumes prior completion of CompTIA A+, Network+, and Server+
    - Considered one of the most difficult CompTIA exams
    - Emphasizes correct command structure and output recognition
    - Requires more than basic conceptual understanding
  - Linux Complexity
    - Multiple ways to perform the same task
    - Modifier order affects output
    - Hands-on practice is necessary

        - Experience and experimentation are essential
    - Certification Value
        - Advances careers in open-source, cloud, and security
    - Exam Structure
        - Five domains
            - System Management – 23%
                - Topics include boot process, kernel modules, partitioning, mounting, shell operations, virtualization, backup and restore
            - Services and User Management – 20%
                - Topics include files and directory management, local accounts, jobs, software, containers, and systemd
            - Security – 18%
                - Topics include AAA, firewalls, OS and account hardening, cryptography, compliance
            - Automation, Orchestration, and Scripting – 17%
                - Topics include automation, Python, Git, Infrastructure as Code, shell scripting, AI
            - Troubleshooting – 22%
                - Topics include system configuration and hardware, storage, networking, and performance issues
    - Exam Objectives
        - 29 total objectives
        - Objectives may combine multiple concepts in scenario-based questions
        - Lessons are structured to follow educational logic, not exam objective order

        - Each lesson title includes the relevant CompTIA objective in parentheses
    - Exam Format
        - Up to 90 questions
        - Multiple choice, multiple select, and performance-based
        - 3 to 5 performance-based simulations expected
        - Passing score is 720 out of 900
        - 80%-85% score on practice tests indicates readiness
    - Voucher Information
        - Vouchers available at store.comptia.org
        - Price approx $370 depending on location
        - 10% discount available through diontraining.com/vouchers
        - Vouchers expire 11 months after purchase
    - Instructor Information
        - Instructor: Jeremiah Minner
        - Instructor background includes cybersecurity, IT, scuba, and martial arts
        - Experience with CompTIA, EC-Council, Cisco, AWS
        - Background in content and lab development for enterprise organizations
    - Learning Tips
        - Closed captions available for each video
        - Playback speed adjustable via video player
        - Downloadable PDF study guide in lesson two
        - Recommended to download and print the CompTIA objectives
    - Student Support
        - Facebook group at facebook.com/groups/diontraining
        - Discord server at diontraining.com/discord
        - Q&A section available on the course landing page

        - Facebook and Discord offer quicker response times due to larger communities
    - Course Strategy
        - Logical order of topic presentation
        - Focus on clear understanding of each objective and subobjective
        - Frequent reinforcement through practice and support

- **Exam Tips**
    - Overview
        - Key guidance and strategies for approaching the CompTIA Linux+ certification exam
    - Exam Tips and Tricks
        - General format and expectations
            - Linux+ exam includes multiple choice, multiple select, and performance-based questions
            - Performance-based questions may require command-line entry or simulated tasks
            - Hands-on knowledge of command syntax and modifiers is critical
            - Recognition of terms, definitions, and concepts from answer choices is required
        - No trick questions
            - Questions are precisely worded
            - Read each question multiple times for clarity
        - Distractors
            - Answer choices may contain red herrings

- Eliminating obvious distractors increases chances of choosing the correct answer

- Keyword emphasis
  - Words in bold, italics, or uppercase must be carefully considered
  - Example
    - MOST, LEAST, BEST
- Knowledge source
  - Answer questions based on course content and CompTIA-approved materials
  - Do not rely on workplace terms or procedures
  - Example
    - Use "allow list" and "block list" instead of workplace terms like "white list" and "black list"
- Selecting the BEST answer
  - Some questions may have multiple correct answers
  - Select the answer that is most accurate in the majority of situations
- Do not fight the test
  - Focus on identifying the key concept being tested
  - Avoid overthinking or inventing edge cases that invalidate questions
  - Example
    - Question
      - "You need to create a compressed archive of an entire directory using the bzip2 method, preserving

the directory structure in a single step. Which of the following commands should you use?"

- Correct answer
  - C. tar -cjvf archive.tar.bz2 /path/to/dir
- Reason
  - The -j switch applies bzip2 compression

- 7 Summary Tips
  - Recognize terms and definitions from choices
  - Understand questions are precisely worded
  - Look out for distractors
  - Focus on emphasized keywords
  - Use official CompTIA knowledge, not workplace experience
  - Choose the BEST answer when in doubt
  - Do not over-analyze or challenge questions unnecessarily

- **100% Pass Guarantee**
  - 100% Pass Guarantee Overview
    - Dion Training offers a 100% pass guarantee for the Linux+ course
    - All risk is assumed by Dion Training
    - Student must actively study and put in effort to pass
  - Course Components
    - Course includes videos
    - Course includes study guide
    - Course includes quizzes
    - Course includes practice exams

- Checkpoint Quizzes
  - Included in each section to assess progress
  - Minimum passing score is 75%
  - Quizzes may be retaken after reviewing lessons and videos
  - Same questions and answer choices appear on each retake
  - A 75% score is required for the quiz to be marked as complete
- Handling Difficult Topics
  - Some topics may appear harder than others
  - Additional study and research are encouraged for difficult topics
- Practice Exams
  - Located at the end of the course
  - Minimum passing score is 75%
  - Practice exams are marked complete upon achieving 75%
  - Score and detailed explanation provided after completion
  - Explanations show why each answer is correct and why it is the best answer
  - Reviewing explanations develops logic and test-taking strategies
- Incorrect Answers
  - Viewed as part of the learning process
  - Used as an opportunity to deepen Linux understanding
- Practice Exam Question Source
  - Practice exams written by Dion Training exam question writers
  - Practice exams do not include actual CompTIA questions
  - CompTIA exam questions are written by different authors
  - Practice exams reflect the format and intent of the real certification exam
- Recommended Course Usage

- Complete all materials in order
- Watch lesson videos
- Take quizzes
- Complete practice exams
- Follow the intended sequence for best results

- Support Contact
  - Email support@diontraining.com for the following
  - Reporting material errors
  - Submitting questions about the content
  - Clarifying concepts
  - Reporting technical issues such as completion tracking
- Certificate of Completion
  - Awarded upon completing all videos
  - Awarded upon passing all quizzes
  - Awarded upon completing all practice exams
  - Certificate qualifies student for the 60-day 100% pass guarantee
  - Certificate must be earned before scheduling the Linux+ exam
- Support Availability
  - Questions about content or concepts may be directed to support@diontraining.com
  - Assistance is available throughout the course

- **Our Lab Environment**
  - Lab Environment Overview
    - Dion Training course includes numerous hands-on labs

- Labs are hosted in a safe, secure, cloud-based environment
- Labs align with video lessons for practical reinforcement

- Lab Access and Content
  - Labs appear throughout the course
  - Labs include interaction with Linux servers
  - Labs include interaction with network infrastructure
  - Labs include interaction with penetration testing systems
  - Labs supplement certification exam preparation and real-world skill development
- Lab Provider
  - Labs designed, built, and operated by CompTIA
  - Labs fully integrated into Dion Training course experience
  - Same labs as CertMaster Lab product from CompTIA store
  - Included with certification course package at Dion Training
- Purpose of Labs
  - Enhance understanding through hands-on experience
  - Simulate real-world job tasks
  - Prepare for Performance-Based Questions (PBQs) on exam
  - Labs provide broader experience than PBQs
  - Lab time ranges from 30 to 90 minutes
  - PBQ time typically ranges from 5 to 10 minutes
  - Labs focus on real-world skill mastery over PBQ performance
  - Mastering labs prepares student to succeed with PBQs
- Technical Support for Labs
  - Labs supported directly by CompTIA
  - For technical issues, go to right-most panel in the lab interface

- Click on the HELP tab
- Use support link to open ticket directly with CompTIA
- Redirected to CompTIA support page
- Submit issues related to lab performance or instructions

- Common Issues and Solutions
  - 95% of lab issues resolved by CompTIA support
  - 5% of issues require Dion Training support
  - Dion Training support helps with concept clarification
  - Dion Training support helps when lab refuses to load
- Dion Training Support Contact
  - Use Discussions or Q&A tab in course platform
  - Email support@diontraining.com
  - Include full-screen screenshot with operating system and browser visible
  - Optionally send screen recording using tools like Loom or Descript
- Lab Launching Best Practices
  - Only one lab can run at a time
  - Lab begins loading in background when launch link appears
  - Navigating to another lesson without exiting causes loading issues
  - Fully launch lab and then use EXIT button in upper corner before leaving
- Importance of Labs
  - Labs are essential to the learning experience
  - Help reinforce course content and build exam readiness
  - Prepare for careers in information technology or cybersecurity
  - Enable experimentation and skill-building in a safe environment
- Final Reminders
  - Use CompTIA support for technical and in-lab issues

- Use Dion Training support for instruction or conceptual issues
- Labs are a tool for learning, practicing, and preparing for success

Linux Concepts

Objective 1.1: *Explain basic Linux concepts*

- **Boot Process**
  - *Linux Boot Process*
    - The sequence of steps a computer follows to start up
    - The four main boot components
      - Preboot Execution Environment (PXE)
        - Optional, used for remote booting
      - Bootloader
        - Loads the OS and prepares the system
      - Initial RAM Disk (initrd)
        - Provides essential drivers and tools before switching to the real root filesystem
      - Kernel
        - The core of the OS, managing system resources
  - Boot Components and Their Roles
    - *Preboot Execution Environment (PXE)*
      - Enables network-based booting instead of local storage (HDD/SSD)
      - Used for deploying operating systems remotely in businesses, schools, and data centers
      - PXE boot process
        - The network card requests an IP address from a DHCP server

    - DHCP provides the location of a TFTP server storing boot files
    - PXE retrieves the bootloader, such as GRUB2, to load the OS installer or system image

- *Bootloader* (GRUB2 - Grand Unified Bootloader v2)
    - Loads the Linux kernel and initrd
    - Interacts with the firmware (BIOS or UEFI) to initialize the boot process
    - Can display a boot menu if multiple OS versions are available
    - Reads boot entries from /boot/grub/grub.cfg
    - Configurations are modified in /etc/default/grub, updated using update-grub
    - Example
        - If multiple kernel versions are installed, GRUB2 allows users to select which one to boot
- *Initial RAM Disk (initrd.img)*
    - A temporary root filesystem loaded before mounting the actual root filesystem
    - Contains essential kernel modules and drivers, such as:
        - Storage controllers
        - Disk encryption modules
        - Filesystem drivers
    - Located in /boot/initrd.img
    - Can be updated using mkinitramfs or dracut, depending on the Linux distribution
    - Example

        - If the root partition is encrypted, initrd contains encryption modules needed to unlock it
    - *Kernel*
        - The core of the operating system
        - Manages hardware, processes, memory, and system resources
        - Mounts the root filesystem and starts system services
        - Configurations are adjusted using sysctl
        - Persistent kernel parameters are stored in /etc/sysctl.conf
        - Changes can be applied immediately with sysctl -p
        - Example
            - To enable IP forwarding, add net.ipv4.ip_forward=1 to /etc/sysctl.conf and apply using sysctl -p
- Linux Boot Process Summary
    - Remote Boot (PXE) [Optional]
        - Starts the bootloader over the network
    - Bootloader (GRUB2)
        - Loads the kernel and initrd
    - Initial RAM Disk (initrd)
        - Provides essential drivers before mounting the real root filesystem
    - Kernel Initialization
        - Mounts the root filesystem and starts system processes

- **System Directories**
    - System Directories
        - Overview of Linux system directories as defined by the Filesystem Hierarchy Standard (FHS)

- *Filesystem Hierarchy Standard (FHS)*
  - Defines how files and directories are organized in Linux
  - Provides standard directory names and use cases
  - Ensures consistency across FHS-compliant Linux distributions
  - Structures the filesystem like a tree, beginning with the root directory
- *Root Directory (/)*
  - Top-level directory from which all other directories branch
  - Represented by a single forward slash (/)
  - Organizes all files and directories within the Linux system
- */boot – Boot Loader Files*
  - Stores files needed to start the operating system
  - Includes
    - /boot/initrd.img
    - /boot/vmlinuz
    - /boot/grub/grub.cfg
  - Initializes hardware and loads the operating system
  - Improper changes can prevent system from booting
- */bin – Essential User Binaries*
  - Contains executable programs required for basic system functionality
  - Used by all users
  - Examples
    - /bin/ls
    - /bin/cp
    - /bin/mv
    - /bin/cat
- */sbin – System Binaries*

- Contains administrative commands used by the root user
- Supports system-level modifications and tasks
- Examples
  - /sbin/fsck
  - /sbin/reboot
  - /sbin/iptables
  - /sbin/mount

- */lib – Shared Libraries and Kernel Modules*
  - Contains essential shared libraries for programs in /bin and /sbin
  - Examples
    - /lib/libc.so.6
    - /lib/ld-linux.so.2
- */dev – Device Files*
  - Contains special files representing hardware devices
  - Allows programs to interact with hardware as files
  - Examples
    - /dev/sda
    - /dev/tty1
    - /dev/null
    - /dev/random
- */proc – Process and System Information*
  - Virtual filesystem providing real-time system and process information
  - Dynamically generated files
  - Examples
    - /proc/cpuinfo
    - /proc/meminfo

- /proc/uptime
- /proc/[PID]/

- */etc – Configuration Files*
  - Stores system-wide configuration files
  - Used to control system behavior and application settings
  - Examples
    - /etc/passwd
    - /etc/shadow
    - /etc/hosts
    - /etc/fstab
    - /etc/ssh/sshd_config
- Summary
  - cd / navigates to the root directory
  - ls displays contents of the directory
  - ls -la shows detailed listing
  - cat displays file contents
  - Demonstrated file locations include
    - /boot
    - /dev
    - /proc
    - /etc

- **User and Application-related Directories**
    - User and Application-related Directories
        - Details of directories used for storing user data, installed applications, dynamic system files, and temporary content in Linux
    - */home – Home Directory*
        - Stores personal files, user settings, and data for each user
        - Each user has a subdirectory named after their username
        - Example
            - /home/jeremiah
        - Contains documents, downloads, and configuration files
        - Users have full control over their own home directories
        - Modifications do not require administrative privileges
    - */usr – User Programs and Application Data Directory*
        - Stores system-wide user-installed programs, libraries, and documentation
        - Files are organized into subdirectories
        - /usr/bin contains application binaries for regular users
        - /usr/sbin contains administrative tools
        - /usr/lib stores shared libraries
        - /usr/share contains documentation and non-executable data
        - Contains software not critical for system startup
        - One of the largest directories in the Linux system
    - */var – Variable Data Directory*
        - Stores files that change frequently
        - Includes log files, email, caches, and databases
        - Examples
            - /var/log for system logs

- /var/mail for email
- /var/www for website data

■ Files persist until manually deleted

○ */tmp – Temporary Files Directory*

■ Stores short-term data created by applications

■ Examples include session files, cached downloads, and temporary logs

■ Files are deleted automatically upon reboot or after a period of inactivity

■ Not intended for storing important or permanent files

○ Summary

■ /home manages user-specific content and configurations

■ /usr holds application binaries and resources for system-wide use

■ /var stores dynamic content like logs and caches

■ /tmp is used for volatile, short-lived application data

- **Server Architectures**

○ *Architecture*

■ Architecture refers to a computer's processor (CPU) design and how it processes data

■ Choosing the right server architecture is critical for reliability, security, and stability in Linux systems

■ The most common Linux server architectures

- x86 (32-bit)
  - Older architecture with memory limitations
- x86_64/AMD64 (64-bit)

        - Powerful, widely supported, and supports large amounts of RAM
    - AArch64 (ARM64)
        - Energy-efficient and scalable for cloud and mobile applications
    - RISC-V
        - Open-source, customizable, and ideal for specialized computing
- Server Architectures and Their Characteristics
    - *x86 (32-bit) Architecture*
        - Developed by Intel, widely used in the 1990s and early 2000s
        - Processes data in 32-bit chunks (4 bytes at a time)
        - Memory limitation
            - Can only access up to 4GB of RAM
        - Still used in older machines, legacy systems, and embedded devices
        - Example
            - An old Linux server running on an Intel Pentium 4 processor
    - *x86_64 / AMD64 (64-bit) Architecture*
        - 64-bit extension of x86, introduced by AMD in 1999 and later adopted by Intel
        - Processes data in 64-bit chunks, improving performance
        - Can theoretically address 16 exabytes (EB) of RAM, but practical limits depend on OS and hardware

- Linux systems running AMD64 architecture typically support up to 128 TB of RAM
- Most modern Linux servers use x86_64 due to its power and widespread compatibility
- Example
  - Enterprise-grade Linux servers running databases or virtualization workloads

■ *AArch64 (ARM64) Architecture*

- 64-bit version of Advanced RISC Machine (ARM) processors
- Designed for power efficiency and scalability
- Popular in mobile devices, cloud servers, and energy-saving data centers
- Many newer Linux servers use AArch64 to reduce power consumption while maintaining performance
- Example
  - Raspberry Pi 4 running Ubuntu Server

■ *RISC-V (Open-Source Architecture)*

- Reduced Instruction Set Computing – 5th iteration
- Open-source CPU design – allows for customized processors
- Unlike x86 and ARM, which are proprietary, RISC-V is free to modify
- Gaining popularity in research, embedded systems, and specialized Linux servers
- Example
  - Used in edge computing devices and AI accelerators for optimized performance in low-power environments

- Key Takeaways
  - Server architecture defines how a processor handles data and determines the best hardware for a Linux server
  - x86 (32-bit) is an older architecture with 4GB RAM limitations, mostly used in legacy systems
  - x86_64/AMD64 (64-bit) is widely used in modern servers, supports large memory, and is high-performance
  - AArch64 (ARM64) prioritizes energy efficiency and scalability, making it popular for cloud computing and mobile applications
  - RISC-V is an open-source, customizable architecture gaining traction in AI accelerators, IoT, and specialized computing

- **Linux Distributions**
  - *Linux*
    - An open-source operating system created by Linus Torvalds in 1991
    - A Linux distribution (distro) is an operating system built on the Linux kernel, including
      - System utilities
      - Libraries
      - Software packages
      - A package management system
    - Linux distributions are categorized based on their package management systems
      - RPM-based
        - Uses the Red Hat Package Manager (RPM) format
      - dpkg-based

        - Uses the Debian package format (.deb)
- *Package Management Systems*
    - Package management systems ensure proper installation, upgrading, and removal of software
    - They also handle software dependencies, ensuring required libraries are installed
- RPM-Based Linux Distributions
    - Key Features
        - Uses the Red Hat Package Manager (RPM) format
        - Package management tools
            - dnf (previously yum)
            - zypper (for openSUSE)
        - Preferred in enterprise environments due to stability, security, and structured release cycles
        - Some distributions, like Red Hat Enterprise Linux (RHEL), require a paid subscription for support
    - Examples of RPM-Based Distributions
        - Red Hat Enterprise Linux (RHEL)
            - Commercial distribution, offers official support
        - Fedora
            - Upstream testing ground for RHEL, latest software updates
        - CentOS Stream
            - Rolling preview of RHEL (replacing traditional CentOS)
        - AlmaLinux & Rocky Linux
            - Free, stable, and binary-compatible alternatives to RHEL
        - openSUSE

        - Uses zypper for package management, known for enterprise and desktop use
    - Package Installation in RPM-Based Systems
        - Using dnf (Fedora, RHEL, CentOS, AlmaLinux, Rocky Linux)

```bash
sudo dnf install firefox
```

        - Using zypper (openSUSE)

```bash
sudo zypper install firefox
```

- dpkg-Based Linux Distributions
    - Key Features
        - Uses the Debian package format (.deb)
        - Package management tools
            - apt (Advanced Package Tool)
                - Fetches and installs software automatically
            - dpkg
                - Manually installs .deb files
        - Extensive repositories and strong community support
        - More user-friendly and frequently updated
    - Examples of dpkg-Based Distributions
        - Debian
            - Highly stable, used as the foundation for many other distros
        - Ubuntu

- Popular, user-friendly, with frequent updates and a large software ecosystem
- Linux Mint
  - Designed for desktop users, similar to Windows UI
- Kali Linux
  - Specialized for penetration testing and cybersecurity

■ Package Installation in dpkg-Based Systems

- Using apt (Ubuntu, Debian, Linux Mint, Kali Linux)

```bash
sudo apt install firefox
```

- Using dpkg (manually installing a .deb file)

```bash
sudo dpkg -i firefox.deb
```

○ Key Differences Between RPM-Based and dpkg-Based Distributions

Feature	RPM-Based Distributions	dpkg-Based Distributions
Package Format	`.rpm`	`.deb`
Main Package Manager	`dnf`, `yum`, `zypper`	`apt`, `dpkg`
Common Distributions	RHEL, Fedora, CentOS Stream, AlmaLinux, Rocky Linux, openSUSE	Debian, Ubuntu, Linux Mint, Kali Linux
Update Model	Structured, stable releases (enterprise-focused)	Frequent updates, community-driven
Software Repositories	Vendor-supported and enterprise-focused	Extensive community-supported repositories
Target Audience	Enterprise, commercial use	General users, developers, cybersecurity professionals

○ Key Takeaways

- Linux distributions are categorized into RPM-based and dpkg-based systems
- RPM-based distributions use the .rpm format with package managers like dnf and zypper
    - Commonly used in enterprises due to stability and structured releases
    - RHEL requires paid subscriptions for official support
- dpkg-based distributions use the .deb format with package managers like apt and dpkg
    - More user-friendly, extensive repositories, strong community support
    - Ubuntu, Linux Mint, and Debian are popular desktop choices
- Both package managers ensure efficient software installation, updates, and dependency resolution

- **Graphical User Interface (GUI)**
    - *Linux Graphical User Interface (GUI)*
        - Includes windows, icons, and menus for interaction
        - Unlike Mac and Windows, Linux allows users to choose different graphical environments
        - The appearance, functionality, and user experience depend on
            - Display system
            - Window manager
            - Display manager
    - Key Components of the Linux GUI
        - *X Server (also called X11 or X Window System)*

- Manages graphical displays on Linux
- Acts as a middle layer between applications and the display
- Handles window rendering, keyboard input, and mouse movements
- Supports network transparency, allowing remote application display using X forwarding over SSH
    - Example
        - Running LibreOffice on an office workstation but displaying it on a home laptop
- Drawbacks of X Server
    - Outdated design introduces security vulnerabilities and performance inefficiencies
    - Many distributions still use Xorg, an implementation of X Server, for software compatibility

- *Wayland* (X Server Alternative)
    - A modern replacement for X Server, designed for better security and efficiency
    - Advantages of Wayland
        - Direct communication between applications and display reduces lag and glitches
        - Improved security by isolating applications from one another
        - More efficient rendering improves graphics performance
    - Challenges of Wayland
        - Not all applications support Wayland (e.g., some screen recording and remote desktop tools)

    - Many distributions, like Ubuntu and Fedora, offer both Wayland and X Server (Xorg)

- *Window Managers*
    - Window managers control how windows move, resize, and interact on the screen
    - Two types of window managers
        - *Floating Window Managers*
            - Users can freely move and resize windows
        - *Tiling Window Managers*
            - Windows are automatically arranged in a grid layout
    - Examples of Window Managers
        - Mutter
            - Floating window manager for GNOME desktop environment
        - KWin
            - Window manager for KDE Plasma, highly customizable with advanced effects
- *Display Managers*
    - Provide a graphical login screen and session control
    - They start the user's desktop environment or window manager
    - Examples of Display Managers
        - GDM (GNOME Display Manager)
            - Used in GNOME-based systems (Ubuntu, Fedora GNOME)
        - SDDM (Simple Desktop Display Manager)

        - Default for KDE Plasma (Kubuntu, Fedora KDE Spin)
    - Key Takeaways
        - Linux GUI is flexible, allowing users to choose different graphical environments
        - X Server (X11) has been the traditional display system but is being replaced by Wayland
            - X Server supports remote display via SSH forwarding but has security vulnerabilities
        - Wayland improves security and efficiency by simplifying communication with the display
        - Window managers define how windows behave (floating vs. tiling)
        - Display managers handle user login and session control

- **Software Licensing**
    - *Software Licensing*
        - Determines how software can be used, modified, and shared
        - Linux follows Free and Open Source Software (FOSS) principles, allowing users to customize, modify, and distribute software freely
        - Understanding Free Software, Open Source Software, Proprietary Software, and Copyleft Software helps users make informed decisions
    - Key Types of Software Licensing
        - *Free Software*
            - Provides users with full freedom to use, modify, share, and distribute
            - "Free" means freedom, not price
            - Emphasizes user rights and ethical software usage

- Example
  - GNU/Linux distributions like Debian and Trisquel

- *Open Source Software*
  - Allows access to source code for transparency, collaboration, and security
  - Encourages community-driven innovation and security improvements
  - Not all Open Source Software aligns with Free Software principles
    - Example
      - Linux is both Free and Open Source, but Google Chrome includes proprietary components
  - Some Open Source Software may include restrictions such as Digital Rights Management (DRM) or proprietary add-ons
- *Proprietary Software*
  - Restricts user access to source code
  - Users cannot modify, share, or distribute the software
  - Often requires a paid license and agreement to strict terms of use
  - Users depend on the vendor for updates, bug fixes, and support
  - Examples
    - Windows, macOS, Adobe software
- *Copyleft Software*
  - Ensures that modified versions of Free Software remain Free Software
  - Requires that any modified versions be shared under the same original license
  - GNU General Public License (GPL) is a common Copyleft license

- Example
  - Linux kernel (GPL-licensed), which requires all modified versions to remain open-source
- Contrast with Permissive Licenses (e.g., Apache Web Server)
  - Permissive licenses allow modifications to become proprietary
  - Example
    - Apache Web Server code can be used in proprietary applications

- Key Takeaways
  - Software licensing determines how software can be used, modified, and distributed
  - Free Software promotes complete user freedom
  - Open Source Software focuses on transparency and collaboration, but may not fully protect user freedoms
  - Proprietary Software restricts access to source code and user modifications
  - Copyleft Software guarantees that software remains Free and Open Source for future users

Shell Operations

Objective 1.5: *Given a scenario, manage a Linux system using common shell operations*

- **User and Session Environmental Variables**
  - Overview
    - Variables in Linux store important data that the system, shell, and applications rely on to function properly
    - Environment variables define session settings and user-specific information
    - User and session environmental variables control aspects like the logged-in user, home directory, shell type, and command prompt appearance
    - Bash variables use a $ sign when referenced (e.g., $USER)
  - Key User and Session Environmental Variables
    - USER Variable
      - Identifies the current logged-in user
      - Example
        - echo $USER prints the current username
      - Applications use this variable to customize settings for the logged-in user
    - HOME Variable
      - Stores the path to the user's home directory
      - Example
        - echo $HOME prints the path (e.g., /home/jeremiah)

- Used by applications to store user-specific files and configurations
- Analogy
  - The HOME directory is like a personal bedroom in a shared house

■ SHELL Variable

- Indicates the current command-line interpreter being used
- Example
  - echo $SHELL prints the shell path (e.g., /bin/bash)
- Common shell options
  - Bash, Zsh, Fish, Csh
- Analogy
  - The SHELL variable is like the language interpreter for commands entered into the terminal

■ PS1 Variable (Prompt String 1)

- Defines the appearance and format of the command prompt
- Example: echo $PS1 prints the current prompt format
- Common PS1 escape sequences
  - \u → Displays current username
  - \h → Displays hostname of the machine
  - \w → Displays current working directory
  - \$ → Displays "$" for regular users, "#" for root users
- Example Command Prompt Output

```
jeremiah@linuxserver:/home/jeremiah/project$
```

  - jeremiah → Username
  - linuxserver → Hostname

        - /home/jeremiah/project ⟶ Current working directory
        - $ ⟶ Normal user prompt (root user would see #)
    - Key Takeaways
        - Environment variables store important session settings and user details
        - $USER ⟶ Stores the currently logged-in username
        - $HOME ⟶ Stores the user's home directory path
        - $SHELL ⟶ Indicates the active command-line interpreter
        - $PS1 ⟶ Controls the format of the terminal command prompt
        - These variables help Linux apply the correct environment settings for each user session automatically

- **System and Execution Environmental Variables**
    - Overview
        - Environmental variables store essential system settings that help Linux manage programs and graphical output
        - System and execution environmental variables direct where the system looks for commands and where graphical applications appear
        - Two key system and execution environmental variables
            - PATH ⟶ Helps Linux locate executables for commands
            - DISPLAY ⟶ Directs graphical output to the correct screen or session
    - Key System and Execution Environmental Variables
        - PATH Variable
            - Specifies directories that store executable programs
            - Example: echo $PATH prints the directories where Linux looks for commands

- Typical PATH output
  - /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
- How PATH works
  - When a user enters a command (e.g., ls), Linux searches directories in PATH order to find the correct executable
  - If the executable is found, it runs; if not, Linux returns "command not found"
- Customizing the PATH variable
  - To add a custom directory (~/scripts) to the PATH for the current session
    - export PATH=$PATH:~/scripts
  - Updated PATH variable
    - /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:~/scripts
  - Making PATH changes permanent
    - Add the export PATH command to shell startup files like ~/.bashrc or ~/.zshrc

- DISPLAY Variable
  - Specifies which screen or session should receive graphical output
  - Example
    - echo $DISPLAY prints the active display session
  - Common DISPLAY values
    - :0.0 → Local display session (first screen on the primary X Server)
    - localhost:10.0 → Remote session forwarded via SSH
  - Customizing the DISPLAY variable

- To direct applications to a second screen (:1), run
  - export DISPLAY=:1
- Then launch an application like Firefox
  - firefox &
- Ampersand (&) runs the program in the background, allowing the user to continue using the terminal

- Key Takeaways
  - Environmental variables define how the system finds programs and directs graphical output
  - $PATH ⟶ Lists directories that store executable programs
    - Adding directories to PATH allows users to run scripts and commands without specifying full file paths
  - $DISPLAY ⟶ Specifies the screen or session where graphical applications should appear
    - Incorrect DISPLAY settings can cause graphical applications to fail to open
  - Both PATH and DISPLAY variables can be modified temporarily or permanently for convenience and efficiency

- **Paths**
  - Overview
    - Paths in Linux function like addresses that tell the system where to locate files and directories
    - Absolute paths provide the full location of a file or folder, starting from a fixed reference point
    - Two key absolute paths

- / (Root Directory) ⟶ The highest level of the filesystem
- ~ (Home Directory) ⟶ The current user's personal directory

- Key Absolute Paths
  - *Root Directory (/)*
    - The top-most directory in the Linux filesystem
    - Every file and directory originates from /
    - Paths that start with / are absolute paths
    - Example
      - /usr/local/bin ⟶ This means
        - Start at / (root)
        - Go into the usr directory
        - Then into local
        - Then into bin
  - *Home Directory (~)*
    - A shortcut to the logged-in user's personal home directory
    - Each user has a unique home directory inside /home/
    - Example
      - If the user is sam
        - ~ = /home/sam
        - cd ~ takes sam to /home/sam
    - Shortcut usage
      - ~/Documents ⟶ The user's Documents folder
      - ~/scripts/myscript.sh ⟶ The myscript.sh file inside scripts

- Key Differences Between / and ~

Feature	Root Directory (/)	Home Directory (~)
Scope	**Universal for the entire system**	**Unique to each user**
Purpose	**Holds all system files and user directories**	**Stores a user's personal files and settings**
Example Path	`/usr/local/bin`	`~/Documents` (which expands to `/home/username/Documents`)
Navigation Command	`cd /` → Moves to the root directory	`cd ~` → Moves to the home directory

Directory	Description	Example (if logged in)
`/`	Root directory for all users	N/A
`~`	User-specific home directory	Depends on the logged-in user
`/home/jon`	Home directory for jon	If jon is logged in
`/home/jake`	Home directory for jake	If jake is logged in
`/home/jeremiah`	Home directory for jeremiah	If jeremiah is logged in
`/home/jesse`	Home directory for jesse	If jesse is logged in

- Key Takeaways
  - Absolute paths always start from a fixed reference point (/ for the system, ~ for the user)
  - / is the root directory, containing all system files, user home directories, and applications
  - ~ represents the home directory of the currently logged-in user
  - Understanding absolute paths helps users navigate the Linux filesystem efficiently

- **Environment Configurations**
    - Environment configurations in Linux
        - Text files that define shell session behavior
        - Shell is a command line interpreter
        - Shell session begins when terminal is opened and ends when closed
    - Bash (Bourne-Again Shell)
        - Commonly used shell and command line interpreter
    - Shell setup using environment configurations
        - Configuration files identify user and apply personalized settings
        - Files processed in the following order
            - .bash_profile
            - .bashrc
            - .profile
    - .bash_profile
        - Used for login shells
        - Loads user-specific environment variables and startup commands
        - Example
            - export JAVA_HOME="/usr/lib/jvm/java-11-openjdk"
                - Sets JAVA_HOME for Java-based applications
        - Example
            - export PATH="$PATH:/opt/new_app/bin"
                - Adds /opt/new_app/bin to the PATH for executing third-party apps
        - Can be configured globally using /etc/skel
            - Used for setting default .bash_profile for new users

- .bashrc
  - Used for interactive, non-login shells
  - Read when a user opens a new terminal or spawns a subshell
  - Contains settings for every shell interaction
  - Commonly includes aliases, functions, prompt customizations, and environment variables
  - Example
    - alias ll="ls -la"
      - Creates a shortcut command ll to run ls -la
- .profile
  - More generic configuration file
  - Read by many POSIX-compliant shells
  - POSIX-compliant shells adhere to Portable Operating System Interface standards
    - Ensures consistent behavior across Unix-like systems
    - Example shell
      - Korn Shell (ksh)
  - Used when .bash_profile is not present
  - Suitable for defining environment variables and startup commands across different shells
  - Example
    - export MY_APP_HOME="/opt/my_app"
      - Ensures compatibility for login shells beyond Bash
- Summary
  - .bash_profile

- For login shells, sets environment variables and startup commands

- .bashrc
  - For interactive, non-login shells, contains UI settings like aliases and functions
- .profile
  - Generic fallback for POSIX-compliant shells, provides consistent shell behavior

- **Input Redirection**
  - Input and Output in Linux
    - Input is the data or commands provided into the terminal
    - Output is the information the system returns back
    - Input redirection changes where a terminal command receives its input from
  - Types of Input Redirection
    - Standard Input (stdin)
    - Basic Input Redirection <
    - Here-document Redirection <<
    - Here-string Redirection <<<
  - Standard Input (stdin)
    - Standard input is the default way a program receives input
    - Standard input usually comes from the keyboard
    - The cat command without redirection waits for additional input
    - Ctrl-D is used to signal the end-of-file (EOF) through stdin
    - This interactive mode demonstrates stdin

- Basic Input Redirection <
  - The < operator is used to redirect input from a file
  - The command cat < myfile.txt displays the contents of myfile.txt
  - The command mail -s "I Love Linux" support@diontraining.com < message.txt uses message.txt as the body of an email
  - The < operator feeds pre-written data into a command automatically
  - For email example to work, a mail command-line utility and MTA must be installed and configured
- Here-document Redirection <<
  - The << operator is known as a here-document
  - Allows embedding multi-line input directly in a command
  - Requires a delimiter to mark the start and end of the text block
  - Example
    - cat <<EOF

      This is the first line.

      This is the second line.

      EOF
    - In a terminal, the heredoc prompt shows lines preceded by heredoc> until the ending delimiter
    - Delimiters can be embedded into scripts for multi-line data input
    - Variations like <<{string} allow defining custom delimiters
- Here-string Redirection <<<
  - The <<< operator is known as a here-string
  - Allows passing a single string as input to a command
  - Example

        - mysql -u admin -p <<< "DROP DATABASE IF EXISTS testdb; CREATE DATABASE testdb; USE testdb; SOURCE /path/to/setup.sql;"
        - This command sends four SQL commands to the MySQL client
        - -u admin specifies the username
        - -p prompts for a password
        - The string includes multiple SQL commands to reset and configure the database
    - Summary of Input Redirection Operators
        - Standard input receives data from the keyboard when no redirection is used
        - < redirects input from a file
        - << allows multi-line input blocks embedded in scripts or commands
        - <<< provides a single string as command input

- **Output Redirection**
    - Output Redirection
        - Details of how to manage command output in Linux using redirection operators
    - *Standard Output (stdout)*
        - Default output stream for command results
        - Displays output on the terminal screen
        - Example command
            - echo "Hello, world!"
        - Output is sent directly to the terminal via stdout
    - *Basic Output Redirection (>)*

- Redirects standard output to a file
- Overwrites existing file content
- Example command
  - echo Hello, world! > greetings.txt
- Writes “Hello, world!” to greetings.txt
- Existing content in greetings.txt is deleted

- *Append Output Redirection (>>)*
  - Appends output to the end of an existing file
  - Preserves previously recorded content
  - Example command
    - echo "Apache service restarted successfully" >> /var/log/service.log
  - Adds message to bottom of service.log without deleting old entries
  - Useful for log files and accumulating outputs over time
- *Standard Error (stderr)*
  - Separate stream for error messages
  - Allows redirection of error messages independently from regular output
  - Redirect error output using 2>
    - Example
      - systemctl restart apache 2> /var/log/apache_error.log
  - Overwrites apache_error.log with error messages
  - Append error output using 2>>
    - Example
      - systemctl restart apache 2>> /var/log/apache_error.log
  - Adds new error messages to existing apache_error.log content
  - Redirect both stdout and stderr using &>

- Example
  - systemctl restart apache &> /var/log/apache_error.log
- Captures all output and error messages into one log file

- Summary
  - stdout is used for regular output display
  - > overwrites a file with new output
  - >> appends output to an existing file
  - 2> and 2>> manage error messages via stderr
  - &> captures both stdout and stderr into a single file

- **History and Shortcuts**
  - History and Shortcuts
    - The command line in Linux serves as a powerful tool for entering commands and recalling previous work
    - History and shortcuts enhance efficiency by allowing quick recall and execution of commands
  - History Feature
    - Saves a record of all commands typed in current and previous sessions
    - Comparable to a personal diary recording actions and conversations
    - Accessed by pressing the Up Arrow key to view the last command
    - Useful for re-executing complex commands without retyping
  - history Command
    - Displays a list of all commands entered in current and previous sessions
    - Helps troubleshoot issues and verify configurations by reviewing past commands

- Example usage
  - history | grep systemctl
    - To find commands related to system service management
- Provides a numbered list of commands for easy recall and execution
- Example
  - 42 sudo apt update
  - 43 systemctl restart networkmanager.service
  - 44 ls -l /var/log

- ! Operator ("bang")
  - Executes a command from history that starts with a specific prefix or is identified by a number
  - Allows for quick re-execution of commands without full retyping
  - Example
    - !43 to re-execute systemctl restart networkmanager.service
  - Retrieves the most recent command starting with a specified prefix (e.g., !sys)
- !! Operator ("bang bang")
  - Quickly re-runs the last executed command
  - Useful for reissuing commands after a failed attempt or for repeated tasks
- alias Command
  - Creates custom shortcuts for longer or complex commands
  - Tailors workflow to individual needs, enhancing efficiency and reducing error chances
  - Aliases can be session-specific or made persistent across sessions
    - For persistent aliases, add them to ~/.bashrc or ~/.bash_profile for personal use

- For system-wide availability, include in /etc/bash.bashrc or /etc/profile

- Example alias
  - alias restartnm='sudo systemctl restart networkmanager.service' allows typing restartnm instead of the full command

- Summary
  - Linux command line not only accepts commands but also retains a record of them
  - Enhances productivity through history management and shortcut commands
  - The history command archives all commands from current and previous sessions for later review and reuse
  - Shortcut commands (!!, !, and alias) streamline workflow by reducing repetitive typing and minimizing errors during system administration

- **Regex**
  - Definition and Purpose of Regex
    - Regular expressions or regex is a specialized language and essential administrator tool used to describe and match patterns within text
    - Regex is not a command
    - Regex is a series of character-based structures that define patterns used for searching text
    - Regex allows rapid location of specific patterns in large amounts of text
    - Regex is useful for troubleshooting and incident response
    - Regex is used to filter command output and automate text processing tasks

- Analogy
  - Searching for a name on a keychain rack vs. searching through a pile illustrates regex functionality
- Core Regex Syntax
  - *Wildcard character*
    - . matches any single character
    - Example
      - c.t matches cat cot or c-t
  - *Escape character*
    - \ treats the next character as a literal
    - Example
      - cowa.bunga matches only cowa.bunga not cowaabunga or cowa-bunga
  - *Anchors*
    - ^ anchors pattern to beginning of line
    - $ anchors pattern to end of line
    - Examples
      - ^Hello matches lines beginning with Hello
      - world$ matches lines ending with world
  - *Character classes*
    - [a-z] matches any single lowercase letter a to z
    - [0-9] matches any single digit
    - Example
      - [a-z]ox matches fox box and pox
  - *Quantifiers*
    - ? matches zero or one occurrence of the preceding element

- matches zero or more occurrences of the preceding element
- matches one or more occurrences of the preceding element
- {n} matches exactly n occurrences of the preceding element
- Example
  - colou?r matches color and colour but not colouur

■ *Alternation*

- | provides a choice between patterns
- Example
  - cat|dog matches either cat or dog

○ Regex with Command-Line Tools

■ Regex defines patterns but does not search by itself

■ Regex must be combined with tools like grep sed and awk

○ Examples of Regex with Tools

■ *grep*

- Used to search through text files for matching lines
- Command
  - grep '^Error.*failed$' logfile.txt
  - ^ ensures match starts with Error
  - $ ensures match ends with failed
  - matches any characters in between

■ *sed*

- Stream editor used for substitution and transformation
- Command
  - sed 's/^Hello/Hi/' file.txt
  - s initiates substitution
  - / separates command components

        - ^ ensures only lines starting with Hello are affected
    - Regex Demonstration
        - Goal
            - Search for IPv4 address in a file using grep
        - Regex pattern
            - [0-9]\{1,3\}\.{3}[0-9]{1,3}
            - [0-9]{1,3} matches 1 to 3 digits for each octet
            - . matches the literal dot
            - ...{3} repeats the first three octets
            - Final [0-9]{1,3} matches the last octet
        - Full command
            - grep '[0-9]\{1,3\}\.{3}[0-9]{1,3}' searchme.log
        - With extended regex
            - grep -E '([0-9]{1,3}.){3}[0-9]{1,3}' searchme.log
    - Benefits of Regex
        - Enables matching of complex patterns
        - Important for searching and analyzing large datasets
        - Requires use with command-line tools like grep and sed for practical application

- **Search and Extract**
    - Overview
        - Search and extract commands allow users to retrieve and isolate information from files efficiently
        - Three core commands covered

- awk
- grep
- cut

- *awk*
  - Named after its creators: Aho, Weinberger, and Kernighan
  - A programming language for pattern scanning and processing
  - Ideal for structured data such as system logs, tables, or spreadsheets
  - Can perform inline filtering, formatting, and arithmetic operations
  - Supports regular expressions and executes code blocks on matched lines
  - Example command
    - awk '/error/ {print $1, $5}' /var/log/syslog
    - Searches for lines with "error"
    - Prints the first field (date) and fifth field (error message) of each matched line
- *grep*
  - Stands for Global Regular Expression Print
  - Used to search for lines matching a specific pattern
  - Supports regular expressions
  - Case sensitive by default
  - -i option makes search case-insensitive
  - Commonly used in log monitoring and troubleshooting
  - Example command
    - grep -i "fail" /var/log/auth.log
      - Searches for all case-insensitive instances of "fail"
- *cut*
  - Used to extract specific sections or fields from files

        - Useful for CSV or colon-delimited data
        - Requires field delimiter specification with -d
        - Extracts fields with -f option
        - Example command
            - cut -d':' -f1 /etc/passwd
            - Uses colon as delimiter
            - Extracts the first field (usernames) from each line in the file
    - Summary
        - awk
            - Complex text processing
            - Structured data analysis
            - Supports regex and field selection
        - grep
            - Fast pattern matching
            - Supports regex
            - Ideal for quick scans of logs or files
        - cut
            - Extracts columns or fields
            - Useful for delimited data formats

- **Modify and Replace**
    - Modify and Replace Commands
        - Commands like sed and tr enable efficient text modification and replacement directly from the command line

        - Essential for automating updates in configurations, correcting errors, and standardizing data across multiple files
    - sed Command
        - sed (stream editor) modifies and replaces text within files or data streams without opening them in a text editor
        - Processes text line by line, allowing pattern search and actions like substitution, insertion, or deletion
        - Syntax
            - sed [modifier] 'command' file
            - Example
                - sed 's/old_hostname/new_hostname/g' config_file.txt
                    - Replaces "old_hostname" with "new_hostname" in config_file.txt
        - Supports various command modifiers
            - d modifier
                - Deletes lines matching a given pattern
            - n modifier
                - Skips automatic printing, moving directly to the next line
            - p modifier
                - Explicitly prints lines, useful with the -n option to control output
            - s command
                - Performs substitutions with the g modifier to replace all matches in a line
        - Utilizes regular expressions for automating complex editing tasks
    - tr Command

- tr (translate) translates or deletes characters in text
- Reads from standard input and delivers transformed output, ideal for character-level modifications
- Example uses
  - Convert uppercase to lowercase
    - tr 'A-Z' 'a-z' < input.txt > output.txt
  - Remove carriage return characters
    - tr -d '\r' < input.txt > output.txt
- Known for simplicity and speed, particularly effective for consistent and clean text processing

- Key Points
  - sed is used for complex text transformations and can handle regular expressions for detailed pattern matching
  - tr focuses on simple character translations or deletions, useful for quick and straightforward text modifications
- Summary
  - Modify and replace commands, such as sed and tr, are powerful tools in Linux for managing text directly from the command line
  - Sed offers a range of functions for intricate editing needs, while tr provides rapid solutions for character-level changes
  - Both tools are vital for system administrators for efficient text editing, automating updates, and ensuring data consistency across files

- **Sort and Count**
  - Sort and Count Commands

- Linux provides commands such as wc, sort, uniq, and xargs to organize and process data efficiently
- Similar to organizing a messy stack of receipts, these commands help sort and manage cluttered information

- wc (Word Count) Command
  - Counts lines, words, and characters in text
  - Syntax
    - wc [options] file
    - Example
      - wc -l /var/log/syslog counts the number of lines in the syslog file
  - Useful for monitoring file sizes and contents, aiding in system diagnostics and troubleshooting
- sort Command
  - Organizes lines of text based on specified criteria
  - Syntax
    - sort [options] file
    - Example
      - sort -t: -k3,3 -n -r /etc/passwd sorts the passwd file by UIDs in descending order
  - Options include
    - -t
      - Specifies a field delimiter
    - -k
      - Indicates the column to sort by
    - -n

        - Numeric sort
    - -r
        - Reverses the order of sort
- Helps identify anomalies or security risks by arranging data logically

- uniq Command
  - Filters out or reports adjacent, repeated lines in sorted data
  - Syntax
    - uniq [options] [input [output]]
  - Often used with sort
    - sort /var/log/auth.log | uniq -c
      - -c
        - Counts and shows how often each line appears
  - Essential for analyzing log files, summarizing data, and detecting patterns in repetitive information
- xargs Command
  - Constructs and executes command lines from standard input
  - Syntax
    - xargs [options] command
    - Example
      - find /home/logs -type f -name "*.log" | xargs -I {} -n 1 -p -t cp {} /backup/logs/
  - Often paired with find to manage files systematically
  - Options include
    - -I {}
      - Replaces {} with the file name found by the find command
    - -n 1

        - Processes one file per command
    - -p
        - Prompts user for confirmation before executing
    - -t
        - Prints the command before execution
  - Facilitates batch processing tasks, automating file management and system maintenance
- Summary
  - wc provides quick insights into the contents of files
  - sort arranges data making it easier to analyze
  - uniq cleans up sorted data, ensuring uniqueness
  - xargs enables automated tasks based on refined data
  - Together, these commands enhance data management and system monitoring in Linux environments

- **View and Navigate**
  - View and Navigate Commands
    - Commands like head and tail allow previewing sections of text files similar to checking the start or end of a book
    - Commands more and less facilitate scrolling through larger files page by page
    - Commands enable quick access to relevant parts of a file, enhancing efficiency in file management
  - head Command
    - Displays the beginning of a file
    - Useful for quick checks on the header or initial configurations of files

- Default behavior shows the first 10 lines of a file
- Syntax
  - head [options] file
- Example usage for viewing the first 50 lines of a configuration file
  - Command
    - head -n 50 /etc/apache2/httpd.conf
  - The -n option specifies the number of lines to view

- tail Command
  - Shows the end of a file
  - Particularly useful for real-time monitoring of log files
  - Default behavior displays the last 10 lines of a file
  - Syntax
    - tail [options] file
  - Example for real-time monitoring of system logs
    - Command
      - tail -n 20 -f /var/log/syslog
    - The -n option specifies the number of lines
    - The -f option follows the file as it grows
- more Command
  - Allows viewing of a long file one page at a time
  - Classified as a pager
  - Syntax
    - more [options] file
  - Often used with the pipe operator to manage output of other commands
    - Example
      - cat /var/log/syslog | more

- Allows scrolling through output page by page using the spacebar or line by line using the Enter key

- less Command
  - Enhanced pager with capabilities beyond those of the more command
  - Allows backward scrolling and in-file searching
  - Syntax
    - less [options] file
  - Example for interactive file viewing
    - Command
      - cat /etc/passwd | less
    - Navigation and search capabilities make it superior for detailed file examination
- Summary
  - Linux view and navigation commands facilitate efficient file management
  - head and tail are crucial for quick previews of file beginnings and ends
  - more and less provide comprehensive tools for navigating through and analyzing large files

- **Output and Redirect**
  - Output and Redirect Commands
    - Understanding how to use echo, cat, and tee commands is crucial for interacting with the Linux system efficiently
    - These commands help manage data, automate tasks, and provide control over files and processes
  - echo Command
    - Prints specified text to the screen or redirects it to another location

- Syntax
  - echo [options] [string]
  - Displays environment variables and handles user input
  - Example
    - echo $HOME displays the home directory
  - Combined with the read command for interactive scripts
    - read -p "Enter your name: " username followed by echo "Welcome, $username!"
- Used in scripts to dynamically write to configuration files or log events
  - Example
    - echo "$(date): System update completed" >> /var/log/sys_update.log appends a log entry with the date

- cat Command
  - Short for "concatenate," used to display, combine, or redirect the contents of files
  - Syntax
    - cat [options] [file...]
    - Displays file contents
      - cat /etc/passwd
    - Merges multiple files
      - cat log1.txt log2.txt log3.txt > full_log.txt
  - Useful in pipelines, for example with grep to filter contents
    - cat /var/log/syslog | grep "ERROR"
- tee Command
  - Writes output to both the terminal and a file simultaneously
  - Syntax

- command | tee [options] [file]
- Commonly used to log command output while observing it
  - ping -c 4 google.com | tee network_log.txt
- Supports appending to files rather than overwriting
  - ping -c 4 google.com | tee -a network_log.txt

■ Enables monitoring and logging for troubleshooting or analysis

- Advanced usage for monitoring SSH
  - journalctl -u sshd -f | tee /var/log/ssh_activity.log | grep "Failed password"

○ Summary

■ echo is versatile for displaying information, capturing user input, and facilitating script automation

■ cat excels in displaying and combining file contents, making it a powerful tool for managing text data

■ tee is essential for both displaying command output live and logging it for record-keeping

■ Together, these commands strengthen your capability to manage and process data effectively in Linux environments

- **Math and Format**

○ Math and Format Commands

■ Focus on commands printf and bc in Linux for formatting output and performing mathematical operations

■ printf enables control over how text and numbers appear, optimizing readability and structure

- bc performs precise arithmetic operations, including handling of decimals and complex calculations

- printf Command
  - Used for formatting output in Linux
  - Provides precise control over the presentation of text and numbers
  - Supports structured output using format specifiers
  - Example of creating a new line or inserting variable values using format specifiers
    - Command
      - printf "User: %s\nHome Directory: %s\nShell: %s\n" "$USER" "$HOME" "$SHELL"
    - Formats and presents user information in a structured way
    - %s acts as a placeholder for string values
    - \n creates a new line after each piece of information
    - Variables like $USER, $HOME, and $SHELL fetch and display user-specific data
    - Output example
      - User
        - username
      - Home Directory
        - /home/username
      - Shell
        - /bin/bash
- bc Command
  - Known as the basic calculator
  - Handles advanced arithmetic and floating-point calculations

- Useful for precise mathematical computations in scripting and automation
- Example for calculating average CPU load with two decimal precision
  - Command
    - echo "scale=2; (2.45 + 3.67 + 4.89) / 3" | bc
  - Maintains accuracy up to two decimal places
  - The scale=2 setting ensures two decimal places in the output
  - Arithmetic operation calculates the average of three values
  - Pipe operator (|) passes the expression to bc for computation
  - Output
    - 3.67

- Summary
  - The printf command is essential for generating structured, readable outputs for reports and logs
  - The bc command offers advanced capabilities for precise arithmetic operations, crucial for system analysis and resource management
  - Together, these commands support efficient data processing, system monitoring, and automation in Linux environments

- **System and Scripts**
  - Overview
    - System and scripts commands help users gather system information and apply script-based environment changes
    - Two key commands discussed
      - uname
      - source

- *uname*
  - Used to retrieve system information
  - Displays kernel name, kernel version, system architecture, and more
  - Common options
    - -r shows kernel release
    - -i displays hardware platform
    - -a prints all system details
  - Useful for checking software compatibility or troubleshooting
  - Example
    - echo "System Info: $(uname -a)" >> /var/log/sys_update.log
      - Captures all system information and appends it to a log file before a kernel update
  - Sample output
    - System Info: Linux server01 5.15.0-86-generic #96-Ubuntu SMP Fri Jan 5 11:42:31 UTC 2024 x86_64 x86_64 x86_64 GNU/Linux
- *source*
  - Executes scripts in the current shell session
  - Applies script changes immediately without starting a new shell
  - Useful for updating environment variables, aliases, and configuration settings
  - Prevents session interruptions for active users
  - Example
    - source /etc/profile.d/custom_vars.sh && echo "Environment variables updated successfully."
      - Reloads the environment variable script and confirms success if no errors occur

- Summary
  - uname
    - Gathers system and kernel information
    - Helps verify properties before changes
  - source
    - Reloads scripts and config files in the current shell
    - Applies environment updates without restarting the session

- **Text editors**
  - Purpose of Linux Text Editors
    - Text editors allow creation and modification of files directly in the terminal
    - Used for writing simple notes or editing important system configuration files
    - Essential for Linux administrators managing scripts and troubleshooting
  - Analogy
    - Nano is like a ballpoint pen
    - Vi/Vim is like a fountain pen requiring practice but offering precision and power
  - *Nano Text Editor*
    - Simplest terminal text editor in Linux
    - Suitable for beginners
    - Allows immediate typing upon opening
    - Administrative shortcuts listed at the bottom of the editor
    - Nano Commands

- Ctrl + O writes the file
- Enter confirms the save
- Ctrl + X exits Nano
- Example command to open a file
    - nano /etc/config-file

- *Vi/Vim Text Editor*
    - Vi and Vim are advanced editors with more features
    - Not intuitive and requires understanding of modes
    - Vi/Vim Modes
        - *Command Mode*
            - Default mode upon starting Vim
            - Used for navigation and editing
        - *Insert Mode*
            - Activated by pressing i in Command Mode
            - Allows text entry
            - Exited by pressing Esc
        - *Execute Mode*
            - Also called Ex Mode or Last Line Mode
            - Activated by pressing : in Command Mode
            - Used for saving quitting searching and replacing text
            - Execute Mode Commands
                - :wq saves and quits
                    - w stands for write
                    - q stands for quit
                - :q! quits without saving
                    - ! forces execution without confirmation

- :set hlsearch highlights all matches of last search query
- :s/old/new/g replaces all occurrences of old with new in the current line
- :!ls runs shell commands without leaving the editor
  - Lists files in current directory

- Advantages of Vim
  - Efficient for script editing and configuration management
  - Useful on remote systems without graphical editors
  - Requires memorization and practice
  - Increases speed and effectiveness in text editing

- Summary
  - Text editors provide fast and efficient file editing in Linux terminal
  - Nano is beginner-friendly with on-screen shortcuts
  - Vi/Vim offers advanced features through multiple modes
  - Mastery of Vi/Vim leads to more powerful and efficient text manipulation
  - Familiarity with both editors allows selection based on task needs

Device Management

Objective 1.2: *Summarize Linux device management concepts and tools*

- **Kernel Module Management**
    - Kernel Module Management Commands
        - Focus on commands insmod, modprobe, and rmmod for managing
        - kernel modules in Linux Kernel modules are analogous to car components, enhancing functionality as needed
        - insmod adds modules without checking dependencies, while modprobe handles dependencies, and rmmod removes modules
    - *insmod* Command
        - Used to manually insert a kernel module into the Linux kernel
        - Requires full path to the module file, typically with a .ko suffix
        - Does not handle dependencies, suitable for controlled testing environments
        - Example for loading a custom module
            - Command
                - insmod /lib/modules/$(uname -r)/extra/custom_module.ko
            - Uses $(uname -r) to ensure compatibility with the current kernel version
            - Lacks error handling and dependency checks, recommended for specific situations
    - *modprobe* Command

- A higher-level tool compared to insmod, automatically managing dependencies
- Requires only the module name, enhancing user-friendliness
- Supports multiple flags for extended functionality
  - -a
    - Loads multiple modules simultaneously
  - -r
    - Safely removes modules, managing dependent unloads
  - -f
    - Force-loads a module, used cautiously for version mismatches
  - -n
    - Performs a dry run to display actions without executing
  - -v
    - Enables verbose mode for detailed output
  - -s
    - Directs output to system logs
- Example for troubleshooting
  - Command
    - modprobe -r wifi_driver && modprobe -v wifi_driver
  - Removes and reloads wifi_driver, useful for error monitoring

- *rmmod* Command
  - Simplest method for removing a kernel module
  - Does not check for dependencies, requiring caution
  - If module is in use or has dependencies, its removal might fail, demanding careful handling

        - Example for removing a USB-related module
            - Command
                - rmmod usb_storage
            - Potential failure if the module is still in use, may need process termination or device unmounting before retry
    - Summary
        - insmod is suitable for inserting modules in specific, controlled situations due to its lack of dependency management
        - modprobe is preferred for regular module management tasks, offering robust dependency handling and error tracking
        - rmmod is used cautiously for removing modules, especially when modprobe -r offers a safer alternative by checking dependencies
        - Understanding these commands helps in managing hardware drivers, optimizing system performance, and troubleshooting Linux systems effectively

- **Module Information and Dependency**
    - Module Information and Dependency Handling
        - Linux provides commands to manage kernel modules similarly to organizing tools in a toolbox
        - Key commands include depmod, lsmod, and modinfo
    - *depmod* Command
        - Analyzes kernel module dependencies and generates the dependency map
        - Ensures correct module loading order by creating or updating the modules.dep file

- Syntax
  - sudo depmod
- To verify dependencies of a module
  - cat /lib/modules/$(uname -r)/modules.dep | grep usbcore
  - Uses $(uname -r) to identify the correct kernel version
  - Uses grep to filter output for specific modules (e.g., usbcore)
- Output format
  - <module_path>: <dependency1> [<dependency2> ...]
  - Example
    - kernel/drivers/usb/storage/usb-storage.ko: kernel/drivers/usb/core/usbcore.ko
  - Ensures modules like usbcore are loaded before dependent modules like usb-storage

- *lsmod* Command
  - Displays all currently loaded kernel modules
  - Output includes module name, size, and other modules that use it
  - Syntax
    - lsmod
  - Helps identify
    - Conflicting modules (e.g., multiple drivers for one device)
    - Unnecessary modules that can be unloaded
    - Unauthorized modules that may indicate a security threat
- *modinfo* Command
  - Provides detailed information about a specific kernel module
  - Syntax
    - modinfo [module_name]

        - Sample output fields
            - filename
                - Module file path
            - description
                - Summary of function
            - author
                - Developer or maintainer
            - license
                - Type of license (e.g., GPL)
            - depends
                - Lists required modules
            - parm
                - Optional configurable parameters
        - Used for
            - Debugging
            - Verifying compatibility
            - Understanding module behavior before changes
    - Summary
        - depmod creates the dependency list so modules load in proper order
        - lsmod shows which modules are currently active and their usage
        - modinfo reveals detailed metadata about a module
        - These tools assist in ensuring system stability, improving performance, and troubleshooting module-related issues in Linux environments

- **Hardware Information Commands**
    - Purpose of Hardware Information Commands

- Provide insight into system components such as BIOS CPU and memory
- Help administrators with audits troubleshooting and performance tuning

- *dmidecode* Command
  - Retrieves hardware information from the DMI table
  - Includes BIOS motherboard processor and memory details
  - Example command
    - sudo dmidecode -t cpu
  - Output example
    - Socket Designation: CPU 1
    - Manufacturer: Intel
    - Version: Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz
    - Core Count: 8
    - Thread Count: 16
    - Max Speed: 5100 MHz
  - Useful for verifying processor specs and ensuring software compatibility
- *lscpu* Command
  - Displays structured CPU information including architecture and cache sizes
  - Extended information available using
    - lscpu -e
  - Output columns
    - CPU shows logical processor number
    - NODE shows NUMA node
    - SOCKET shows physical CPU socket
    - CORE shows core number within socket
    - L1d:L1i:L2:L3 shows cache sizes

- ONLINE shows whether processor is active

- Helps with
  - Process scheduling
  - Workload distribution
  - NUMA balancing
  - High-performance computing tuning
  - Detecting offline CPUs
  - Configuring CPU affinity

- *lsmem* Command
  - Displays memory structure and allocation at block level
  - Example command
    - lsmem
  - Output columns
    - RANGE shows start and end address of memory block
    - SIZE shows memory block size
    - STATE indicates whether the block is online
    - REMOVABLE shows whether block is hot-swappable
    - BLOCK identifies memory block number
  - Useful for
    - Viewing memory layout
    - Diagnosing hardware issues
    - Checking availability and removability of memory
- Summary
  - dmidecode provides BIOS CPU and memory details from system firmware
  - lscpu offers processor architecture core and cache data for tuning performance

- lsmem breaks down memory block allocation and availability
- Combined use of these commands supports system audits diagnostics and optimization

- **Hardware Inventory Commands**
  - Purpose of Hardware Inventory Commands
    - Used to identify hardware components installed on a Linux system
    - Provide information about internal and external devices
    - Help in determining system specifications for management and troubleshooting
  - Commands Covered
    - lspci
    - lsusb
    - lshw
  - *lspci* Command
    - Lists all PCI-connected hardware devices
    - Includes graphics cards network adapters sound cards and more
    - Structured output can be viewed using -tv option
    - Command example
      - lspci -tv
    - Output elements
      - -[0000:00]- root PCI bus
      - +- shows hierarchy and connection
      - 00.0 01.0 etc are device addresses
      - Text after each address describes device and manufacturer

- Provides an organized view of PCI device structure
- *lsusb* Command
  - Lists connected USB devices
  - Includes keyboards mice flash drives webcams and more
  - Command example
    - lsusb
  - Output elements
    - Bus and Device number indicate USB location
    - ID indicates vendor and product ID
      - Example
        - ID 046d:c52b
        - 046d is Logitech manufacturer ID
        - c52b is product code
    - Device name identifies connected USB device
  - Useful for checking device recognition and USB troubleshooting
- *lshw* Command
  - Provides a detailed report on all hardware components
  - Includes data from both lspci and lsusb plus memory CPU storage and network
  - Command example
    - sudo lshw
  - Truncated summary available with -short option
    - sudo lshw -short
  - Output elements
    - H/W path shows hardware hierarchy
    - Device column lists associated device files

        - Example
            - /dev/sda
    - Class column categorizes hardware type
        - Examples
            - system memory processor disk network
    - Description provides human-readable names
- Summary
    - Hardware inventory commands offer visibility into system components
    - lspci lists PCI devices with structural hierarchy using -tv
    - lsusb lists USB peripherals with detailed IDs and names
    - lshw gives complete hardware overview including memory storage CPU
    - lshw -short provides a simplified and quick reference view

- **System Monitoring and Diagnostics Commands**
    - System Monitoring and Diagnostics
        - Commands like dmesg, lm_sensors, and ipmitool help monitor hardware health, detect issues, and troubleshoot problems
        - Serve as a system's "dashboard" by providing real-time and historical insights
    - *dmesg* Command
        - Displays kernel ring buffer messages including hardware detection, driver activity, and system errors
        - Useful for diagnosing boot issues, crashes, or hardware failures
        - Syntax: dmesg [options]
            - -c

- Clears the buffer after displaying
- -f
  - Filters by facility (e.g., kernel, drivers)
- -l
  - Filters by severity level (e.g., err)
- -e
  - Displays timestamps
- -L
  - Enables color-coded output
- -H
  - Formats output for human readability
- -h
  - Displays help menu

- Example
  - dmesg -l err -H filters only error messages in a readable format
  - Output Example
    - [ERROR] CPU0: Temperature too high!
    - [ERROR] NVMe disk not detected, check connections.
    - [ERROR] USB device failed to enumerate.

- *lm_sensors*
  - Package for monitoring temperatures, voltages, and fan speeds
  - Includes commands like sensors and watch sensors
  - sensors
    - Displays current readings
  - watch sensors
    - Updates sensor readings every two seconds

- Example output
  - coretemp-isa-0000
    - Displays CPU package and core temperatures
  - nct6776-isa-0290
    - Shows fan speeds and voltage readings
- Helps detect overheating, fan failures, and abnormal voltage levels

- *ipmitool* Command
  - Interacts with IPMI (Intelligent Platform Management Interface) for remote server management
  - Syntax
    - ipmitool [options] <command> [command-specific arguments]
    - ipmitool sensor
      - Displays temperature, fan speed, and voltage data
    - ipmitool chassis status
      - Shows power status of the system
    - ipmitool power cycle
      - Remotely reboots the system
    - ipmitool lan print
      - Displays IP configuration of management interface
  - Provides remote diagnostics and monitoring for enterprise systems
- Summary
  - dmesg reveals system and kernel messages critical for error tracking and debugging
  - lm_sensors offers real-time thermal, fan, and voltage monitoring to prevent hardware failures

        - ipmitool enables remote access to hardware diagnostics and control for server environments
        - Combined, these commands support proactive system health monitoring and rapid issue resolution in Linux systems

- **Initrd Management**
    - *Initrd Management*
        - Initrd (initial RAM disk) is a temporary filesystem used during boot to load essential drivers before mounting the real root filesystem
        - Two key commands
            - mkinitrd (legacy tool)
            - dracut (modern standard)
    - *mkinitrd* Command
        - Creates a manual initrd image
        - Syntax
            - mkinitrd [options] <initrd-image> <kernel-version>
        - Options
            - --preload=<module>
                - Loads specified module early in the boot process
            - --with=<module>
                - Adds a specific kernel module
            - -f
                - Forces creation of a new image even if one exists
            - --nocompress
                - Prevents compression to simplify inspection
        - Example

- mkinitrd --preload=ext4 --with=usb-storage -f --nocompress /boot/initrd-5.15.0.img 5.15.0
    - Preloads ext4, includes usb-storage, forces image creation, and disables compression

- *dracut* Command
    - Modern alternative that automates initramfs creation
    - Syntax
        - dracut [options] <initramfs-image> <kernel-version>
    - Dynamically includes only needed drivers and components
    - Options
        - --force
            - Overwrites existing initramfs image
        - --add-drivers "<driver1> <driver2>"
            - Explicitly adds drivers
        - --omit-drivers "<driver1> <driver2>"
            - Excludes unnecessary drivers
        - --no-compress
            - Disables compression for easy debugging
    - Example
        - dracut --force --add-drivers "xfs nvme" --no-compress /boot/initramfs-5.15.0.img 5.15.0
            - Adds xfs and nvme drivers, forces image recreation, and keeps the image uncompressed
- Summary
    - mkinitrd allows manual initrd image creation with customization options

        - dracut automates initramfs creation with advanced dependency management and modular design
        - Both tools ensure proper module loading during the Linux boot process, crucial after kernel or hardware updates

- **Custom Hardware**
    - Overview
        - Linux supports a wide range of custom hardware from low-power embedded systems to high-performance GPUs
        - Key areas
            - Embedded Systems and GPU Use Cases
    - Embedded Systems
        - Small, specialized computers performing limited tasks efficiently
        - Built for reliability and low power consumption
        - Commonly found in
            - Smart home devices
            - Medical equipment
            - Industrial automation
        - Run lightweight Linux distributions or Real-Time Operating Systems (RTOS)
            - RTOS ensures consistent, immediate responses to inputs
            - Ideal for systems requiring strict timing and stability
        - Examples of embedded system use
            - Traffic light controllers
            - Heart rate monitors
        - Play a key role in Internet of Things (IoT) devices

- Power applications like security cameras and self-driving cars

- GPU Use Cases
  - GPUs (Graphics Processing Units) enable high-performance computing
  - Designed for parallel processing, unlike CPUs
  - Used in
    - Video editing
    - Artificial Intelligence (AI)
    - Scientific simulations
    - Cryptocurrency mining
  - Monitoring Tool: nvtop (NVIDIA/AMD Top)
    - Real-time dashboard for GPU performance metrics
    - Displays
      - Power consumption (watts)
      - Temperature (°C) Utilization percentage
      - Memory usage
      - Running GPU processes with
        - User
        - PID
        - Workload type (compute/graphics)
        - Allocated GPU memory
  - Helps administrators
    - Identify high-resource processes
    - Ensure efficient GPU operation
- Summary
  - Embedded Systems prioritize efficiency and stability, often using RTOS or lightweight

- Linux GPUs support high-demand workloads using parallel processing
- Linux tools like nvtop provide real-time insights into GPU performance
- Together, these capabilities showcase Linux's versatility in managing both low-power embedded devices and high-powered computational hardware across diverse environments and applications

Storage Management

Objective 1.3: *Given a scenario, manage storage in a Linux system*

- **Partition Information Commands**
  - Partition Information Commands
    - Partition commands help explore and understand system storage layout
    - Key tools
      - lsblk
      - blkid
  - *lsblk* Command
    - Displays a structured view of block devices, partitions, and mount points
    - Syntax
      - lsblk [options]
      - -f
        - Displays filesystem type, UUID, and mount points
    - Default output presents a tree-like layout of drives and their partitions
    - Output columns include
      - NAME
        - Device name with parent-child relationships
      - FSTYPE
        - Filesystem type (e.g., ext4, ntfs)
      - LABEL
        - Partition label (if set)
      - UUID

- Universally Unique Identifier for consistent recognition
- MOUNTPOINT
  - Directory where the partition is mounted
- Example command
  - lsblk -f
- Example output
  - sda
    - sda1 ext4 boot UUID=1234-5678-ABCD-EFGH /boot
    - sda2 ext4 root UUID=A1B2-C3D4-E5F6-G7H8 /
    - sda3 swap UUID=8765-4321-DCBA-HGFE
  - sdb
    - sdb1 ntfs data UUID=1122-3344-5566-7788 /mnt/data

- *blkid* Command
  - Displays detailed metadata about block devices
  - Syntax
    - blkid [options] [device]
  - Provides information useful for persistent mounting in /etc/fstab
  - Key fields
    - UUID
      - Ensures device recognition across reboots
    - TYPE
      - Filesystem type (e.g., ext4, ntfs)
    - LABEL
      - User-defined name for the partition
  - Example command
    - blkid /dev/sda2

        - Example output
            - /dev/sda2: UUID="A1B2-C3D4-E5F6-G7H8" TYPE="ext4" LABEL="root"
    - Summary
        - lsblk gives a high-level structural view of storage devices and their mount points
        - blkid provides metadata such as UUIDs, filesystem types, and labels for identifying and managing partitions
        - Used together, they offer a complete picture of system storage, supporting effective and consistent disk management

- **Partition Management Commands**
    - *fdisk and gdisk*
        - Used to create, delete, and manage partitions on a disk
        - fdisk supports MBR partition tables
        - gdisk supports GPT partition tables
        - Generic syntax
            - fdisk [options] <device>
        - Example usage
            - sudo fdisk /dev/sda
        - Interactive sequence

```
n   # Create a new partition
p   # Choose a primary partition
1   # Partition number 1
[Enter]   # Accept default start sector
[Enter] or +10G   # Accept default or set partition size
w   # Write changes to disk
```

- Interactive prompt options
  - n
    - Creates a new partition
  - d
    - Deletes a partition
  - p
    - Prints the current partition table
  - w
    - Writes changes
  - q
    - Exits without saving
- Example output
  - The partition table has been altered.
- gdisk provides similar functionality with support for GUID-based partition IDs

- *parted*
  - Supports interactive and direct command-line partitioning
  - Used for selecting disks, creating, resizing, and removing partitions
  - Generic syntax
    - parted [device] [command] [options]
  - Example usage
    - sudo parted /dev/sdb mkpart primary ext4 1MiB 20GiB
  - parted commands
    - select
      - Chooses a disk

- mkpart
  - Creates a partition
- print
  - Displays the partition table
- resizepart
  - Resizes a partition
- rm
  - Removes a partition
- quit
  - Exits the tool

- Example output for print

```
Model: ATA SSD (scsi)
Disk /dev/sdb: 500GB
Partition Table: gpt
Number  Start   End     Size    Type     File system
1       1MiB    20GiB   20GiB   primary  ext4
```

- Allows non-destructive resizing of partitions

- *growpart*
  - Used to expand existing partitions
  - Does not create or delete partitions
  - Useful in cloud and virtual machine environments
  - Generic syntax
    - growpart <device> <partition_number>
  - Example usage
    - sudo growpart /dev/sda 1
  - Example output
    - CHANGED: partition=1 start=2048 old: [50G] new: [100G]

        - Expands partition to use all available unallocated space
    - Summary
        - fdisk and gdisk manage partition tables using an interactive interface
        - parted enables advanced partitioning with non-destructive resize options
        - growpart expands existing partitions to use unallocated space
        - Tools support efficient storage management across local and cloud-based systems

- **Redundant Array of Independent Disks (RAID)**
    - *Redundant Array of Independent Disks (RAID)*
        - RAID combines multiple physical drives into logical units
        - Enhances performance redundancy or both
        - RAID configurations include
            - Striping for speed in RAID 0
            - Mirroring for protection in RAID 1
            - Parity for error detection and recovery in RAID 5 and RAID 6
    - *mdadm*
        - Primary Linux tool for managing RAID arrays
        - Supports creation modification monitoring of RAID devices
        - Syntax
            - mdadm [mode] [options] <RAID device>
        - Options
            - --create initializes a new RAID array
            - --manage adds or removes disks from an existing array
            - --monitor continuously watches array for failures

- --verbose provides detailed output
- Create RAID 1 array
  - sudo mdadm --create --verbose /dev/md0 --level=1 --raid-devices=2 /dev/sdb /dev/sdc
    - /dev/md0 is the RAID device
    - --level=1 specifies RAID 1 (mirroring)
    - --raid-devices=2 assigns two physical drives
- Save configuration
  - sudo mdadm --detail --scan >> /etc/mdadm/mdadm.conf
  - sudo update-initramfs -u
- Format and mount array
  - sudo mkfs.ext4 /dev/md0
  - sudo mkdir /mnt/raid
  - sudo mount /dev/md0 /mnt/raid
- Add to /etc/fstab
  - /dev/md0 /mnt/raid ext4 defaults 0 0
    - Enables automatic mounting at boot
- Monitor RAID status
  - sudo mdadm --monitor --verbose /dev/md0
  - Example output
    - mdadm: Monitor mode set for /dev/md0
    - mdadm: Disk /dev/sdb failed in /dev/md0 – marking as faulty.
    - mdadm: Rebuild started on spare /dev/sdc.

- */proc/mdstat*
  - Virtual file showing real-time RAID status

- Displays RAID level device list and synchronization state
- Check RAID status
  - cat /proc/mdstat
- Example output

```
Personalities : [raid1]
md0 : active raid1 sda1[0] sdb1[1]
      500000 blocks [2/2] [UU]
unused devices: <none>
```

  - md0 is an active RAID 1 array
  - sda1 and sdb1 are active disks
  - [UU] indicates both disks are healthy
  - [U_] indicates a failed disk

- Summary
  - RAID increases speed redundancy or fault tolerance depending on configuration
  - mdadm is used for setup monitoring and management
  - /proc/mdstat gives a snapshot of RAID health and activity
  - RAID 0 uses striping
  - RAID 1 uses mirroring
  - RAID 5 and RAID 6 use parity for fault tolerance

- **Basic Physical Volume Management**
  - Physical Volume Management Overview
    - Physical Volumes (PVs) are foundational components in Logical Volume Management (LVM)

- PVs must be initialized before inclusion in Volume Groups (VGs)
- Key commands
  - pvcreate, pvdisplay, pvs, and pvscan

- *pvcreate* Command
  - Initializes a raw storage device or partition as a Physical Volume
  - Syntax
    - pvcreate /dev/<device>
  - Example
    - pvcreate /dev/sdb
    - Output
      - Physical volume "/dev/sdb" successfully created
  - Prepares the device for inclusion in a Volume Group (VG)
- *pvdisplay* Command
  - Provides detailed information about a specific Physical Volume
  - Syntax
    - pvdisplay /dev/<device>
  - Example Output
    - PV Name: /dev/sdb
    - VG Name: <not assigned>
    - PV Size: 100.00 GiB
    - Allocatable: yes
    - PE Size: 4.00 MiB
    - Total PE: 25599
    - Free PE: 25599 Allocated PE: 0
    - PV UUID: JZ9h3D-Ty8N-M3qX-27Lg-K6N8-xb2N-LMZ7G5
  - Shows UUID, allocation details, and extent size

- Indicates whether PV is assigned to a VG

- *pvs* Command
  - Displays a summarized table of all Physical Volumes
  - Syntax
    - pvs
  - Example Output
    - PV VG Fmt Attr PSize PFree
    - /dev/sda1 vg01 lvm2 a-- 200.00G 50.00G
    - /dev/sdb lvm2 a-- 100.00G 100.00G
  - Provides quick overview
    - size, free space, volume group assignment
- *pvscan* Command
  - Scans all available disks for Physical Volumes
  - Syntax
    - pvscan
  - Example Output
    - PV /dev/sda1 VG vg01 lvm2 [200.00 GiB / 50.00 GiB free]
    - PV /dev/sdb lvm2 [100.00 GiB]
    - Total: 2 [300.00 GiB] / in use: 1 [200.00 GiB] / in no VG: 1 [100.00 GiB]
  - Confirms system recognition of all PVs
  - Useful after adding new storage
- Summary
  - pvcreate initializes a device for use as a Physical Volume in LVM
  - pvdisplay shows detailed attributes of a Physical Volume
  - pvs gives a high-level summary of all PVs in table format

        - pvscan detects all Physical Volumes on the system
        - These tools collectively ensure proper initialization, monitoring, and detection of PVs in LVM systems for effective storage management in Linux environments

- **Modifying and Removing Physical Volumes**
    - Modifying and Removing PVs
        - LVM allows resizing, relocating, or removing Physical Volumes (PVs) based on changing storage needs
        - Key commands
            - pvresize, pvmove, and pvremove
    - *pvresize* Command
        - Resizes a Physical Volume after underlying partition or disk has been resized
        - Syntax
            - pvresize /dev/<device>
        - Example
            - pvresize /dev/sdb
        - Example Output
            - Physical volume "/dev/sdb" changed
            - 1 physical volume(s) resized / 0 physical volume(s) not resized
        - Updates LVM to reflect the new size of the PV
    - *pvmove* Command
        - Moves data from one PV to another within the same Volume Group
        - Useful for disk replacement or reorganization
        - Syntax

- pvmove /dev/<source> /dev/<destination>
- Example
- pvmove /dev/sdb /dev/sdc
- Real-time progress output
- /dev/sdb: Moved: 10.0%
- /dev/sdb: Moved: 45.0%
- /dev/sdb: Moved: 100.0%
- Ensures data is safely transferred before removing or repurposing the old PV

- *pvremove* Command
- Removes a Physical Volume from
- LVM Must be removed from any Volume Group before use
- Syntax
- pvremove /dev/<device>
- Example
- pvremove /dev/sdb
- Example Output
- Labels on physical volume "/dev/sdb" successfully wiped
- Erases LVM metadata so the disk can be reused or removed

- Summary
- pvresize updates LVM to match the new size of a disk or partition
- pvmove transfers data between PVs with real-time progress
- pvremove deletes a PV from LVM after it is disassociated from a Volume Group
- These commands provide dynamic, flexible storage management within LVM systems in Linux environments

- **Basic Volume Group Management**
    - Volume Group Management Overview
        - Volume Groups (VGs) organize physical storage devices into flexible storage pools
        - Key commands
            - vgscan, vgcreate, vgdisplay, vgs
    - *vgscan* Command
        - Scans the system for existing volume groups
        - Syntax
            - vgscan
        - Example output
            - Reading all physical volumes. This may take a while...
            - Found volume group "vg_data" using metadata type lvm2
        - Purpose
            - Detects all available volume groups, especially after adding disks or restoring backups
    - *vgcreate* Command
        - Creates a new volume group from one or more physical volumes
        - Syntax
            - vgcreate <volume_group_name> <physical_volume>
        - Example
            - vgcreate vg_projects /dev/sdb1
        - Example output
            - Volume group "vg_projects" successfully created
        - Purpose
            - Sets up a new volume group for managing logical volumes

- *vgdisplay* Command
  - Provides detailed information about a specific volume group
  - Syntax
    - vgdisplay <volume_group_name>
  - Example
    - vgdisplay vg_projects
  - Example output highlights
    - VG Name
      - Name of the volume group
    - VG Size
      - Total storage size
    - Cur PV
      - Number of current physical volumes
    - Act PV
      - Number of active physical volumes
    - Alloc PE / Size
      - Allocated physical extents and their size
    - Free PE / Size
      - Free physical extents and their size
  - Purpose
    - Displays space allocation, volume status, and physical volume details
- *vgs* Command
  - Provides a summarized overview of all volume groups
  - Syntax
    - vgs

- Example output

VG	#PV	#LV	#SN	Attr	VSize	VFree
vg_data	1	2	0	wz--n-	500.00g	100.00g
vg_projects	1	0	0	wz--n-	100.00g	100.00g

- Key Columns
  - VG
    - Volume Group name
  - #PV
    - Number of physical volumes
  - #LV
    - Number of logical volumes
  - #SN
    - Number of snapshots
  - Attr
    - Attributes such as writable and resizable
  - VSize
    - Total volume size
  - VFree
    - Available free space
- Purpose
  - Quick, comparative view of volume groups and their status

- Summary
  - vgscan detects existing volume groups on the system
  - vgcreate establishes new volume groups from physical volumes

        - vgdisplay provides in-depth configuration details for a specific volume group
        - vgs offers a brief summary of all volume groups for quick monitoring
        - Mastering these commands helps administrators efficiently manage and scale storage systems in Linux environments

- **Modifying and Removing Volume Groups**
    - Overview
        - Volume group management allows flexible adjustments to storage resources
        - Commands used: vgextend, vgchange, vgexport, vgimport, vgremove
    - *vgextend*
        - Adds a physical volume to an existing volume group
        - Syntax
            - vgextend <volume_group_name> <physical_volume>
        - Example
            - vgextend vg_projects /dev/sdc1
    - *vgchange*
        - Activates or deactivates a volume group
        - Syntax
            - vgchange [options] <volume_group_name>
        - Example (activate)
            - vgchange -a y vg_data
        - Example (deactivate)
            - vgchange -a n vg_data
    - *vgexport*

        - Prepares a volume group for transfer to another system
        - Marks the group as inactive and exported
        - Syntax
            - vgexport <volume_group_name>
        - Example
            - vgexport vg_archive
    - *vgimport*
        - Imports an exported volume group on a new system
        - Makes it available to LVM
        - Syntax
            - vgimport <volume_group_name>
        - Example
            - vgimport vg_archive
    - *vgremove*
        - Deletes a volume group with no remaining logical volumes
        - Syntax
            - vgremove <volume_group_name>
        - Example
            - vgremove vg_temp
    - Summary
        - vgextend increases group capacity by adding new physical volumes
        - vgchange toggles access to volume groups
        - vgexport and vgimport allow safe migration of volume groups
        - vgremove deletes unused volume groups after clearing logical volumes

- **Basic Logical Volume Management**
    - Logical Volume Management (LVM) Overview
        - Logical Volumes (LVs) are customizable storage areas within Volume Groups (VGs)
        - Commands used
            - lvcreate, lvdisplay, lvs
    - *lvcreate* Command
        - Creates a logical volume from an existing volume group
        - Syntax
            - lvcreate --name <logical_volume_name> --size <size> <volume_group_name>
        - Example
            - lvcreate --name lv_data --size 20G vg_projects
            - Breakdown
                - --name lv_data
                    - Names the new logical volume lv_data
                - --size 20G
                    - Allocates 20 gigabytes of space
                - vg_projects
                    - Specifies the volume group in which the LV is created
        - Purpose
            - Allocates and reserves space for specific storage tasks
    - *lvdisplay* Command
        - Provides detailed information about a specific logical volume
        - Syntax

- lvdisplay <logical_volume_path>
- Example
  - lvdisplay /dev/vg_projects/lv_data
- Purpose
  - Confirms properties and status of a logical volume

- *lvs* Command
  - Displays a summary of all logical volumes in the system
  - Syntax
    - lvs
  - Example Output

LV	VG	Attr	LSize
lv_data	vg_projects	-wi-a-----	20.00g
lv_logs	vg_data	-wi-a-----	50.00g

  - Key Columns
    - LV
      - Logical Volume name
    - VG
      - Volume Group name
    - Attr
      - Volume attributes
        - w: Writable
        - i: Inherited permissions
        - a: Active
    - LSize
      - Logical volume size

        - Purpose
            - Provides a high-level view of logical volumes and their usage
    - Summary
        - lvcreate initializes a new logical volume with specified name, size, and VG
        - lvdisplay shows detailed information about a logical volume
        - lvs provides a quick, summarized overview of all logical volumes
        - These tools help configure, monitor, and manage logical storage efficiently in Linux systems using LVM

- **Modifying and Removing Logical Volumes**
    - Overview
        - Logical volume management allows resizing activation and deletion of volumes based on changing storage needs
        - Key commands include lvchange, lvresize, lvextend, and lvremove
    - *lvchange*
        - Changes status or attributes of a logical volume
        - Used to activate or deactivate a volume
        - Syntax
            - lvchange [options] <logical_volume_path>
        - Example
            - lvchange -a y /dev/vg_projects/lv_data
                - -a y activates the volume
                - -a n would deactivate the volume
    - *lvresize*
        - Changes the size of a logical volume (increase or decrease)
        - Shrinking requires shrinking the filesystem first to avoid data loss

- Syntax
  - lvresize --size <new_size> <logical_volume_path>
- Example
  - lvresize --size 15G /dev/vg_projects/lv_data
    - Sets the logical volume size to 15 gigabytes
- Important
- Always shrink the filesystem first before reducing logical volume size
- Back up data before resizing to prevent corruption

- *lvextend*
  - Specifically used to increase the size of a logical volume
  - Requires separate filesystem resizing after extension
  - Syntax
    - lvextend --size <additional_size> <logical_volume_path>
  - Example
    - lvextend --size +5G /dev/vg_projects/lv_data
      - Adds 5 gigabytes to the current volume size
  - Follow with
    - resize2fs for ext-based filesystems
    - xfs_growfs for XFS filesystems
- *lvremove*
  - Permanently deletes a logical volume from the system
  - Frees up space in the volume group
  - Syntax
    - lvremove <logical_volume_path>
  - Example
    - lvremove /dev/vg_projects/lv_data

        - Prompts for confirmation before deletion
        - Once confirmed, volume is permanently removed
    - Summary
        - lvchange activates or deactivates a logical volume
        - lvresize adjusts the size of a volume (increase or decrease)
        - lvextend increases the volume size and must be followed by filesystem resizing
        - lvremove deletes a logical volume from the system
        - These tools help maintain efficient and adaptable storage in a Linux environment

- **Static and System Mount Information**
    - Overview
        - Linux uses three files to manage mounted filesystems
        - /proc/mounts shows real-time mounted devices
        - /etc/mtab lists mounted devices as tracked by user-space tools
        - /etc/fstab defines static mount points for automatic mounting
    - */proc/mounts*
        - Displays current, live mount information from the kernel
        - Not editable by users
        - Always reflects the real-time state of the system
        - Command
            - cat /proc/mounts
        - Example output
            - /dev/sda1 / ext4 rw,relatime,data=ordered 0 0

        - /dev/sda1
            - Block device
        - /
            - Mount point
        - ext4
            - Filesystem type
        - rw
            - Read-write access
        - relatime
            - Updates access times on reads
        - data=ordered
            - Journaling option
        - 0 0
            - Not backed up by dump, no fsck at boot
- */etc/mtab*
    - Traditionally maintained by mount and umount commands
    - Historically writable but now often linked to /proc/self/mounts
    - Slightly less accurate than /proc/mounts but useful to scripts
    - Command
        - cat /etc/mtab
- */etc/fstab*
    - Specifies static filesystems to mount at startup
    - Defines six fields per line
        - Block device
        - Mount point
        - Filesystem type

- Mount options
- Dump setting (0 = do not dump)
- fsck order (1 = root, 2 = others, 0 = none)

- Command to mount all fstab entries
  - mount -a
- Example entry
  - /dev/sdb1 /mnt/data ext4 defaults 0 2

- Summary
  - /proc/mounts is a live, kernel-maintained view of current mounts
  - /etc/mtab is a traditional user-space view, often linked to /proc/self/mounts
  - /etc/fstab defines which filesystems should be mounted at boot
  - mount -a can trigger mounting of all defined entries in /etc/fstab without reboot

- **Dynamic Mounting and Automounting**
  - Dynamic Mounting and Automounting
    - Linux supports dynamic mounting and automounting for managing filesystem access
    - Commands include
      - mount
      - umount
      - autofs
  - *mount*
    - Used to connect a filesystem to a directory (mount point)

- Makes filesystem contents accessible under that directory
- If the directory does not exist use mkdir -p /mnt/mydrive
- -p creates parent directories if not present
- Generic syntax
  - mount [OPTIONS] <SOURCE> <TARGET>
- Example
  - mount -t ext4 /dev/sdb1 /mnt/mydrive
    - -t ext4 specifies filesystem type
    - /dev/sdb1 is the source device
    - /mnt/mydrive is the mount point
- Output
  - mount: /mnt/mydrive: mounted on /dev/sdb1

- *umount*
  - Used to safely detach a mounted filesystem
  - Prevents data corruption and ensures consistency
  - Required before physically removing a device like a USB stick
  - Generic syntax
    - umount [OPTIONS] <TARGET|SOURCE>
  - Example
    - umount /mnt/mydrive
      - /mnt/mydrive is the mount point being unmounted
  - Output
    - umount: /mnt/mydrive unmounted
- *autofs*
  - Automates mounting and unmounting filesystems on demand
  - Uses a daemon to attach on access and detach after idle time

- Configuration involves /etc/auto.master and a map file
- Start service
  - systemctl start autofs
- Enable service at boot
  - systemctl enable autofs
- Output example
  - Created symlink
    /etc/systemd/system/multi-user.target.wants/autofs.service →
    /usr/lib/systemd/system/autofs.service
- Enables automatic mounting on access and unmounting after inactivity

- Summary
  - mount attaches a filesystem to a directory for use
  - umount detaches a filesystem to prevent corruption
  - autofs automates mounting and unmounting based on activity

- **Performance & Accessibility Mount Options**
  - Performance and Accessibility Mount Options
    - Linux mount options control filesystem access and system performance
    - Options include ro, rw, remount, noatime, and nodiratime
  - *ro and rw*
    - Control read and write access to a filesystem
    - ro sets filesystem as read-only to prevent modifications
    - rw allows both read and write access
    - Syntax
      - mount -o ro|rw <SOURCE> <TARGET>

- Example
  - mount -o ro /dev/sdb1 /mnt/mydrive
    - -o ro makes /mnt/mydrive read-only
    - Files can be read but not changed

- *remount*
  - Changes mount options without unmounting the filesystem
  - Useful for switching between read-write and read-only modes with minimal disruption
  - Syntax
    - mount -o remount,<OPTIONS> <TARGET>
  - Example
    - mount -o remount,ro /mnt/mydrive
      - Remounts the existing mount point /mnt/mydrive as read-only

- *noatime*
  - Disables updates to file access time on reads
  - Improves performance by reducing disk writes during read-heavy operations
  - Syntax
    - mount -o noatime <SOURCE> <TARGET>
  - Example
    - mount -o noatime /dev/sdb1 /mnt/mydrive
      - Prevents access time updates for files
      - Enhances disk I/O performance

- *nodiratime*
  - Disables access time updates for directories only

        - Maintains file-level access time updates
        - Reduces disk writes from frequent directory lookups
        - Syntax
            - mount -o nodiratime <SOURCE> <TARGET>
        - Example
            - mount -o nodiratime /dev/sdb1 /mnt/mydrive
                - Speeds up directory access by avoiding unnecessary updates
    - Summary
        - ro restricts filesystem to read-only mode
        - rw enables read-write access
        - remount changes access modes without unmounting
        - noatime disables file access time updates to reduce writes
        - nodiratime disables directory access time updates for performance

- **Security & Restriction Mount Options**
    - Overview
        - Linux mount options enhance security by restricting filesystem behavior
        - Common security options include nodev, nosuid, and noexec
    - *nodev*
        - Prevents special device files on a filesystem from being treated as actual devices
        - Files such as /dev/tty or /dev/sda placed on the filesystem are treated as plain files
        - Prevents unauthorized access to hardware or system functions

- Syntax
  - mount -o nodev <device> <mountpoint>
- Example
  - sudo mount -o nodev /dev/sdb1 /mnt/safe
    - /mnt/safe will treat device files as regular files
    - Enhances security by disabling device file functionality

- *nosuid*
  - Prevents execution of files with set-user-ID (SUID) or set-group-ID (SGID) bits
  - Blocks files from gaining elevated privileges when run
  - Reduces risk of hidden privilege escalations
  - Syntax
    - mount -o nosuid <device> <mountpoint>
  - Example
    - sudo mount -o nosuid /dev/sdb1 /mnt/no-suid
      - Files in /mnt/no-suid will not execute with elevated privileges
- *noexec*
  - Prevents execution of any binaries or scripts located on the filesystem
  - Useful for storage areas where execution should not be allowed
  - Enhances protection against unauthorized or malicious code
  - Syntax
    - mount -o noexec <device> <mountpoint>
  - Example
    - sudo mount -o noexec /dev/sdb1 /mnt/lockdown
      - Executables in /mnt/lockdown will not be allowed to run

- Summary
  - nodev disables device file functionality in a mount point
  - nosuid disables SUID and SGID privilege escalations
  - noexec disables execution of files on the filesystem
  - These options help limit damage from malicious programs or unauthorized activities

- **Network Mounts**
  - Network Mounts
    - Network mounts connect computers to share files as if they were local
    - Common types include NFS (Network File System) and SMB (Server Message Block or Samba)
  - *NFS (Network File System)*
    - Designed for Linux-to-Linux file sharing
    - Remote directories behave like local storage
    - Ideal for environments using Unix-like systems
    - Syntax
      - mount -t nfs <server>:/<remote-path> <local-mountpoint>
    - Example
      - sudo mount -t nfs 192.168.1.100:/exports/data /mnt/data
        - -t nfs specifies NFS type
        - 192.168.1.100:/exports/data is the server and export path
        - /mnt/data is the local directory where the share appears
    - Common in environments requiring centralized data access for multiple Linux systems

- *SMB (Server Message Block / Samba)*
  - Supports cross-platform file sharing including Windows and Linux
  - Implemented in Linux as Samba
  - Uses CIFS (Common Internet File System) protocol
  - Syntax
    - mount -t cifs //<server>/<share> <local-mountpoint> -o username=<user>,password=<pass>
  - Example
    - sudo mount -t cifs //192.168.1.50/shared /mnt/shared -o username=admin,password=secret
      - -t cifs specifies CIFS protocol for SMB
      - //192.168.1.50/shared is the SMB server and share name
      - /mnt/shared is the local mount point
      - -o username=admin,password=secret provides login credentials
  - Useful in mixed environments where both Windows and Linux users need file access
- Summary
  - Network mounts make remote filesystems appear locally
  - NFS is best for Linux-to-Linux file sharing
  - SMB (Samba) supports file sharing across different operating systems
  - Simplifies collaboration and centralizes file access in networked environments

- **Filesystems Formats**
    - Filesystem Formats
        - Filesystem formats organize and manage data across different Linux environments
        - Key formats
            - ext4, XFS, btrfs, and tmpfs
    - *ext4 Filesystem*
        - Known for reliability and efficiency
        - Command to create
            - mkfs.ext4 /dev/sdXN
        - Supports
            - Large files (up to around 16 terabytes with 4KB block size)
            - Very large filesystems (approaching exabyte scale theoretically)
        - Characteristics
            - Disk-based storage using memory for caching and metadata management
            - Minimal memory restrictions
        - Commonly used in desktop and server environments
    - *XFS Filesystem*
        - Designed for high performance and scalability
        - Command to create
            - mkfs.xfs /dev/sdXN
        - Supports
            - Very large files and volumes, often scaling into petabytes
        - Characteristics
            - Optimized for disk-based storage performance

        - Minimal memory restrictions
    - Ideal for enterprise environments handling large files and heavy workloads
- *btrfs Filesystem*
    - Next-generation filesystem with advanced features
    - Command to create
        - mkfs.btrfs /dev/sdXN
    - Command to inspect layout
        - btrfs filesystem show
    - Supports
        - Snapshots
        - Subvolumes
        - Built-in data integrity checks
    - Characteristics
        - Flexible, dynamic data management
        - Performance varies depending on workload
    - Suited for environments needing incremental backups and advanced data management
- *tmpfs Filesystem*
    - Temporary filesystem that resides entirely in RAM
    - Command to create
        - sudo mount -t tmpfs -o size=512M tmpfs /mnt/tmp
    - Characteristics
        - Extremely fast read and write speeds
        - Size limited by system RAM and set options
        - Data is not persistent across reboots

        - Best for performance-critical tasks requiring temporary storage
    - Summary
        - ext4 offers stability, journaling, and large filesystem support with low memory impact
        - XFS provides high performance and scalability for handling very large volumes
        - btrfs introduces snapshots, subvolumes, and data integrity features for dynamic management
        - tmpfs enables high-speed temporary storage entirely in system memory
        - Together, these filesystems cover a wide range of Linux storage needs from stability to advanced performance optimization

- **Inodes**
    - *Inodes*
        - Inodes store metadata for files in a Linux filesystem
        - Metadata includes ownership permissions and location on disk
        - Inodes do not store file names or contents
    - inode usage
        - Inode is an ID card for each file directory or object in the filesystem
        - Every file requires a unique inode regardless of file size
        - Directories are special files that map filenames to inode numbers
        - Each directory also requires its own inode
    - *inode exhaustion*
        - Total number of inodes is fixed at filesystem creation
        - Determined by disk size and filesystem type

- inode exhaustion occurs when inode limit is reached before disk space is full
- Prevents creation of new files despite available storage space
- Monitoring inode usage
  - Use df -i to display inode usage statistics
  - Example output includes
    - Inodes
    - IUsed
    - IFree
    - IUse%
    - Mounted on

```
Filesystem      Inodes  IUsed   IFree IUse% Mounted on
devtmpfs        481247    591  480656    1% /dev
tmpfs           483248    652  482596    1% /run
/dev/sda1      2560000  75000 2485000    3% /
tmpfs           483248     10  483238    1% /dev/shm
tmpfs           483248     18  483230    1% /run/lock
tmpfs           483248     15  483233    1% /run/user/1000
```

- filesystem creation
  - Inodes are allocated during filesystem creation
  - Example
    - mkfs -t ext4 /dev/sdb1
      - Sets aside inodes for the new ext4 filesystem
  - Custom inode allocation may be needed for large numbers of small files
- file relationships
  - Files and directories are tied to inodes
  - Hard links point multiple filenames to the same inode
  - Use ls -i to display inode numbers of files

- Example output
  - 328883 example.txt
  - 328883 example-hardlink.txt
  - 328950 anotherfile.txt
- example.txt and example-hardlink.txt share the same inode number
- Indicates both refer to the same data on disk

- Summary
  - Inodes store critical metadata for filesystem objects
  - inode exhaustion occurs when all inodes are used even if disk space remains
  - Filesystem creation reserves a fixed number of inodes
  - Hard links connect multiple filenames to the same inode
  - Tools like df -i and ls -i help monitor and understand inode usage

- **Filesystem Utilities**
  - Overview
    - Filesystem utilities allow setup, inspection, repair, and resizing of Linux storage
    - Useful for maintaining filesystem health and adapting storage needs
    - Commands include df, du, fio, mkfs, fsck, xfs_repair, resize2fs, and xfs_growfs
  - *df*
    - Displays used and available disk space
    - df stands for "disk free"

        - df -h shows human-readable format (e.g., GB, MB)
    - *du*
        - Shows disk usage of files and directories
        - du stands for "disk usage"
        - du -sh /path provides summarized human-readable output
        - Useful for identifying which folders consume the most space
    - *fio*
        - Stands for "Flexible I/O Tester"
        - Benchmarks read/write disk performance
        - Simulates workloads like random reads or sequential writes
        - Useful for diagnosing slow storage or evaluating new drives
    - *mkfs*
        - Creates a new filesystem on a partition or disk
        - mkfs -t <type> <device> specifies the filesystem type
        - Example
            - mkfs -t ext4 /dev/sdb1 creates ext4 filesystem
    - *fsck*
        - Stands for "filesystem check"
        - Verifies and repairs filesystem metadata
        - Commonly used after power failures or system crashes
        - Should be used only on unmounted filesystems
        - Example
            - fsck -fy /dev/sdb1 forces a check and auto-confirms fixes
    - *xfs_repair*
        - XFS-specific utility to check and repair issues
        - Used instead of fsck on XFS filesystems

- Fixes superblock errors, orphaned files, and more
- Required if XFS wasn't unmounted cleanly

- *resize2fs*
  - Used to grow or shrink ext2, ext3, or ext4 filesystems
  - Must resize the underlying partition first
  - Example use case
    - Increase storage without reformatting
- *xfs_growfs*
  - XFS equivalent of resize2fs
  - Only supports growing filesystems, not shrinking
  - Simple usage
    - xfs_growfs /mount/point
  - Expands into newly added partition space
- Summary
  - df checks total free/used disk space
  - du pinpoints which directories consume space
  - fio benchmarks read/write performance
  - mkfs creates new filesystems with specified type
  - fsck checks and repairs ext-based filesystems
  - xfs_repair is for checking and repairing XFS
  - resize2fs modifies ext2/3/4 filesystem size
  - xfs_growfs expands XFS into newly allocated space

File and Directory Management

Objective 2.1: *Given a scenario, manage files and directories on a Linux system*

- **Navigation and Directory Management Commands**
  - Navigation and Directory Management Commands
    - *pwd*
      - Displays the full path of the current working directory
      - Syntax
        - pwd
      - Example output
        - /home/admin/Documents/scripts
          - Current location is the scripts directory inside Documents under /home/admin
    - *ls*
      - Lists contents of a directory
      - Syntax
        - ls [options] [directory]
      - Example
        - ls -l
          - Lists files with details such as permissions, owner, size, and last modified date
      - Example output
        - -rw-r--r-- 1 admin admin 2048 Mar 30 14:20 report.txt
        - drwxr-xr-x 2 admin admin 4096 Mar 30 13:05 projects

- Use ls -la to include hidden files
  - Hidden files begin with a dot
  - Example output
    - -rw-r--r-- 1 admin admin 2048 Mar 30 14:20 .bashrc
    - -rw-r--r-- 1 admin admin 2048 Mar 30 14:20 report.txt

- *cd*
  - Changes the current directory
  - Syntax
    - cd [directory]
  - Example
    - cd Documents/projects
      - Navigates into the projects directory inside Documents
  - Shortcut
    - cd ..
      - Moves to the parent directory
  - Shortcut
    - cd ~
      - Returns to the home directory
- *mkdir*
  - Creates a new directory
  - Syntax
    - mkdir [directory-name]
  - Example
    - mkdir logs

- Creates a directory named logs in the current location

- *rmdir*
  - Removes an empty directory
  - Syntax
    - rmdir [directory-name]
  - Example
    - rmdir logs
      - Deletes the empty logs directory
  - Note
    - Only works on empty directories
    - To remove non-empty directories, use rm -r (covered in a different lesson)

- Summary
  - pwd shows current directory path
  - ls lists directory contents and can show details or hidden files
  - cd navigates through directories with shortcuts like cd .. and cd ~
  - mkdir creates new directories
  - rmdir removes empty directories

- **File Operation Commands**
  - File Operation Commands
    - *touch*
      - Creates a new, empty file
      - Syntax

- touch [filename]
- Example
  - touch notes.txt
    - Creates a file named notes.txt in the current directory

- *cp*
  - Copies files or directories
  - Syntax
    - cp [source] [destination]
  - Example
    - cp notes.txt backup.txt
      - Copies contents of notes.txt to backup.txt
- *mv*
  - Moves or renames files
  - Syntax
    - mv [source] [destination]
  - Example
    - mv notes.txt /home/admin/Documents/
      - Moves notes.txt to the Documents directory
- *rm*
  - Removes files or directories
  - Syntax
    - rm [options] [filename or directory]
  - Example
    - rm backup.txt
      - Deletes backup.txt permanently

- Option
    - -i
        - Prompts for confirmation before deletion
        - Example
            - rm -i todo.txt
                - Output
                    - rm: remove regular file 'todo.txt'?
    - Option
        - -f
            - Forces deletion without warnings or prompts
            - Example
                - rm -f todo.txt
    - Option
        - -R
            - Recursively deletes a directory and its contents
            - Example
                - rm -R myfolder/

- Summary
    - touch creates new empty files
    - cp copies files or directories
    - mv moves or renames files
    - rm deletes files or directories permanently

- Use -i to confirm deletions, -f to force, and -R for recursive directory deletion

- **File Information and Search Commands**
  - File Information and Search Commands
    - *file*
      - The file command is used to determine the type of data a file contains
      - Linux does not rely on file extensions to identify file types
      - Syntax
        - file [filename]
      - Example
        - file backup1
      - Example output
        - backup1: ASCII text
      - ASCII text indicates that backup1 is a plain text file
      - Prevents mistakes such as attempting to run a text file as an executable
    - *stat*
      - The stat command provides detailed information about a file or directory
      - Information includes last access time, ownership, size, and permissions
      - Syntax
        - stat [filename]

- Example

    - Access displays the last time the file was opened or read
    - Modify displays the last time the file's contents were changed
    - Change displays the last time metadata such as permissions or ownership were modified
    - The stat output helps administrators track usage and detect suspicious changes

- *find*
    - The find command searches for files or directories in a directory tree
    - Results can be filtered by name, type, permissions, and other attributes
    - Syntax
        - find [path] [options]
    - Example
        - find /var/log -type f -name "*.log" -perm 644
            - -type f limits the search to regular files
            - -name "*.log" filters for files ending in .log

- -perm 644 filters files with specific read/write permissions
- Useful for locating sensitive logs with incorrect permissions

- *locate*
  - The locate command searches using a prebuilt index of system files
  - Syntax
    - locate [options] [search_term]
  - -i option makes the search case insensitive
  - Example
    - locate -i config
  - Returns all files with “config” in the name regardless of capitalization
  - locate is faster than find because it uses an indexed database
  - Useful for quick searches without the need for filtering by type or permissions

- Summary
  - file identifies the data type of a file
  - stat provides file metadata including size, ownership, and timestamps
  - find performs real-time searches based on file properties
  - locate uses an indexed approach for faster queries

- **File Comparison and Usage Commands**
  - File Comparison and Usage Commands
    - *diff*

- The diff command compares the contents of two files line by line
- Useful for tracking changes and identifying configuration mismatches
- Options include
    - -b ignores changes in the amount of whitespace
    - -w ignores all whitespace differences
    - -i makes comparison case-insensitive
    - -t preserves tab characters
    - -c and -u provide contextual formatting
- Syntax
    - diff [options] file1 file2
- Example
    - diff -i -u config_old.txt config_new.txt
- Example output

```
--- config_old.txt 2025-04-01 10:00:00
+++ config_new.txt 2025-04-02 14:12:00
@@ -2,7 +2,7 @@
 server=production
 port=443
-LogLevel=warning
+LogLevel=info
 timeout=30
```

    - --- shows the original file
    - +++ shows the comparison file
    - @@ marks the location of the change
    - - indicates removed lines
    - + indicates added lines

- Lines without symbols are unchanged
- Useful for collaborative environments and version tracking

- *sdiff*
  - The sdiff command displays differences between two files side-by-side
  - Makes visual comparison easier when changes are minimal or scattered
  - Syntax
    - sdiff [options] file1 file2
  - Example
    - sdiff policy_old.txt policy_new.txt
  - Example output
    - min_length=8| min_length=12
    - require_numbers=yesrequire_numbers=yes
  - Symbols in the middle column indicate
    - | difference between files
    - < line only in the first file
    - > line only in the second file
- *lsof*
  - The lsof command lists currently open files and the processes using them
  - Useful for identifying file locks, monitoring usage, and troubleshooting issues
  - Syntax
    - lsof [options] [target]
  - Example

- lsof /var/log/syslog
- Example output

```
COMMAND    PID     USER   FD     TYPE DEVICE    SIZE/OFF NODE  NAME
rsyslogd   456     root   3w     REG  253,0     10560    6543  /var/log/syslog
```

- COMMAND identifies the process
- PID is the Process ID
- USER is the file owner
- FD indicates the file descriptor and mode (e.g., 3w for writing)
- NAME confirms the file path in use

- Summary
  - diff compares file content and shows exact changes with contextual output
  - sdiff presents file differences side-by-side for easier visual analysis
  - lsof reveals which files are open and which processes are using them

- **File Links**
  - Overview
    - File links allow multiple references to a file's data or path
    - ln command creates hard or symbolic links
    - Hard links point to the same inode and data
    - Symbolic links point to the path of the original file
  - *ln Command*
    - Used to create both hard and symbolic (soft) links

- General syntax
  - ln [options] source target
- -s
  - Creates symbolic links
- -f
  - Forces overwrite of destination
- -i
  - Prompts before overwrite
- --backup
  - Makes a backup before overwrite
- -v
  - Provides verbose output

- *Hard Links*
  - Share the same inode and data as the original file
  - Both files are fully interchangeable
  - Changes to one affect the other
  - If the original is deleted, the data persists
  - Created without the -s option
  - Example
    - ln server.log server_backup.log
      - Confirm with ls -li to view identical inode numbers
- *Symbolic Links*
  - Point to the path of the original file
  - Have a separate inode from the target
  - Break if the target is moved or deleted
  - Created using ln -s

        - Example
            - ln -s /etc/nginx/nginx.conf ~/nginxlink.conf
                - Confirm with ls -l showing -> path reference
    - Summary
        - ln is the command to create links
        - Hard links share the same inode and survive deletion of original
        - Symbolic links reference file paths and break if target is missing
        - Use hard links for redundancy and data persistence
        - Use symbolic links for convenient shortcuts to files or paths

- **/dev Device Types**
    - /dev Device Types
        - *Block devices*
            - Block devices manage and process data in fixed-size blocks
            - Used for storage devices such as hard drives, SSDs, and USB flash drives
            - Suitable for managing file systems and large data transfers
            - Found in the /dev directory
            - Example device names
                - /dev/sda
                - /dev/sdb1
            - /dev/sda represents the primary hard disk
            - /dev/sdb1 represents the first partition on a secondary device
            - Example command
                - mkfs.ext4 /dev/sdb1

- Formats /dev/sdb1 with the ext4 file system

- *Character devices*
  - Character devices process data one character at a time in a continuous stream
  - Used for input/output devices such as keyboards, serial ports, and terminals
  - Found in the /dev directory
  - Example device name
    - /dev/tty
  - Represents a terminal interface
  - Example command
    - cat /dev/ttyUSB0
  - Displays data from a connected USB serial device
  - Useful for debugging hardware or reading sensor data
- *Special character devices*
  - Special character devices do not represent physical hardware
  - Perform specific system-level functions
  - Common special character devices
    - /dev/null
    - /dev/zero
    - /dev/urandom
  - /dev/null discards all data sent to it
  - Example command
    - ./noisy-script.sh > /dev/null 2>&1
  - Silences both standard output and error output of the script
  - /dev/zero outputs a continuous stream of zeroes

- Example command
  - dd if=/dev/zero of=dummyfile bs=1M count=10
- Creates a 10MB file filled with zeroes
- /dev/urandom generates random data
- Often used for cryptography and secure password generation
- Example command
  - head -c 16 /dev/urandom | base64
  - Creates a 16-byte random string encoded in base64

- Summary
  - Block devices are ideal for file system operations and storage management
  - Character devices are essential for managing real-time data streams
  - Special character devices assist with system automation, testing, and scripting

Local Account Management

Objective 2.2: *Given a scenario, perform local account management in a Linux environment*

- **Add**
  - *Add*
    - Local account management creates users and groups to manage system access
    - Commands include groupadd, useradd, and adduser
  - *groupadd*
    - Creates new user groups for organizing access control
    - Syntax
      - groupadd [options] groupname
    - Example
      - sudo groupadd engineering
        - Creates a group named engineering
    - Exit codes
      - 0 success
      - 2 incorrect syntax or missing option
      - 4 internal account management file error
      - 9 group already exists
    - Check exit code
    - echo $?
  - *useradd*
    - Creates new user accounts with detailed configuration

- Syntax
  - useradd [options] username
- Example
  - sudo useradd -c "Jane Doe - Engineer" -m -s /bin/bash -e 2025-12-31 -u 1050 jane
    - -c adds comment
    - -m creates home directory
    - -s sets shell to Bash
    - -e sets account expiration date
    - -u sets user ID
    - jane is the username
- View default settings
  - useradd -D

- *adduser*
  - Interactive script for user creation
  - Uses useradd in the background
  - Syntax
    - adduser username
  - Example

```
sudo adduser mark

Adding user `mark' ...
Adding new group `mark' (1003) ...
Adding new user `mark' (1003) with group `mark' ...
Creating home directory `/home/mark' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for mark
Enter the new value, or press ENTER for the default
Full Name []: Mark Johnson
Room Number []:
Work Phone []:
Home Phone []:
Other []:
Is the information correct? [Y/n]
```

        - Suitable for new administrators needing a guided setup
    - Summary
        - groupadd defines new user groups
        - useradd creates accounts with customizable options
        - adduser provides an interactive walkthrough for account creation
        - These tools ensure secure and organized user access in Linux systems

- **Delete**
    - *Delete*
        - Used to safely remove users and groups from a Linux system
        - Commands include userdel, deluser, and groupdel
    - *userdel*
        - Permanently removes a user account from the system
        - Syntax
            - userdel [options] username
        - Example
            - sudo userdel -r emma
                - -r removes the home directory and mail spool
                - Removes the account entry from the system
        - Exit codes
            - 0 success
            - 1 cannot update account files
            - 2 invalid command syntax
            - 6 user does not exist
            - 8 user is currently logged in

- Check exit code
  - echo $?

- *deluser*
  - User-friendly wrapper script to remove users
  - Common on Debian-based systems
  - Syntax
    - deluser [options] username
  - Example
    - sudo deluser emma
      - Removes user account but retains home directory
      - Use --remove-home to also delete the home directory
  - Provides readable output and additional checks
- *groupdel*
  - Deletes a group from the system
  - Syntax
    - groupdel [options] groupname
  - Example
    - sudo groupdel marketing
      - Deletes the marketing group if not assigned as a primary group
  - Exit codes
    - 0 success
    - 2 invalid syntax
    - 6 group does not exist
    - 8 permission denied
    - 10 group is primary for an existing user

        - Check exit code
            - echo $?
        - Example error
            - groupdel: cannot remove the primary group of user 'lisa'
    - Summary
        - userdel removes a user and optionally their files
        - deluser is a simplified interactive tool for removing users
        - groupdel deletes unused groups from the system
        - Exit codes help with command verification and troubleshooting

- **Modify**
    - Modify
        - *passwd command*
            - The passwd command manages a user's password
            - Allows setting, changing, deleting, locking, unlocking, and expiring passwords
            - Syntax
                - passwd [OPTIONS] [USERNAME]
            - Example to delete a user’s password
                - sudo passwd -d samantha
            - Example to lock a user’s account
                - sudo passwd -l samantha
            - Example to unlock a user’s account
                - sudo passwd -u samantha
            - Example to expire a user’s password

    - sudo passwd -e samantha
  - Expiring a password forces the user to change it at the next login
- *chsh command*
  - The chsh command stands for change shell
  - Used to change a user's login shell
  - Useful for assigning a different shell environment
  - Syntax
    - chsh -s [SHELL] [USERNAME]
  - Example to change a user's shell to Zsh
    - sudo chsh -s /bin/zsh samantha
  - The -s option specifies the new shell
  - Shell changes can be verified in the /etc/passwd file
- *groupmod command*
  - The groupmod command modifies an existing group
  - Can change the group name or properties
  - Syntax
    - groupmod [OPTIONS] [GROUPNAME]
  - Example to rename a group
    - sudo groupmod -n devs developers
  - -n specifies the new group name
  - Exit codes
    - 0 indicates success
    - 2 invalid options
    - 6 failure to update the group file
    - 8 user not found
    - 10 group name already exists

- *usermod command*
  - The usermod command modifies user accounts
  - Can change shell, add to groups, set expiration, and more
  - Syntax
    - usermod [OPTIONS] [USERNAME]
  - Example to add a user to a group
    - sudo usermod -aG admin samantha
  - -a appends the group
  - -G specifies the group
  - Example to change a user's shell
    - sudo usermod -s /bin/zsh samantha
  - Example to set an account expiration date
    - sudo usermod -e 2025-12-31 samantha
  - Exit codes
    - 0 indicates success
    - 1 invalid option
    - 2 missing or incorrect argument
    - 6 user does not exist
    - 8 insufficient permissions

- Summary
  - passwd manages user authentication by handling password states
  - chsh changes the shell interface for user login sessions
  - groupmod modifies group settings and structure
  - usermod provides extensive modification capabilities for user accounts

- **Lock**
    - Overview
        - Linux administrators control account access using commands
            - passwd
            - usermod
            - chage
        - These tools help lock/unlock accounts and enforce password policies
        - Ensures appropriate access for the right users at the right time
    - *passwd* Command
        - Used for managing user passwords and account locking/unlocking
        - Lock an account
            - sudo passwd -l rterrance
        - Unlock an account
            - sudo passwd -u rterrance
        - Locks disable the user's password, preventing login
        - Unlocking restores password-based login
        - Useful for temporary access suspension without deleting the account
    - *usermod* Command
        - Modifies user account details, including locking/unlocking
        - Lock an account
            - sudo usermod -L rterrance
        - Unlock an account
            - sudo usermod -U rterrance
        - -L
            - Disables the password, blocking login
        - -U

        - Re-enables login by unlocking the account
    - *chage* Command
        - Manages password expiration and aging policies
        - Syntax
            - chage [OPTIONS] [USERNAME]
        - Set password to expire every 30 days and warn 7 days before
            - sudo chage -M 30 -W 7 rterrance
        - -M
            - Sets maximum password age
        - -W
            - Sets warning period before expiration
        - View password aging info
            - sudo chage -l rterrance
        - Set account expiration date
            - sudo chage -E 2025-12-31 jdoe
        - -E
            - Disables account after the specified date
        - Helps ensure passwords are updated regularly and accounts are not left active indefinitely
    - Summary
        - passwd
            - Locks/unlocks account passwords for login control
        - usermod
            - Enables or disables account access more broadly
        - chage

- Manages password aging, expiration warnings, and account disablement

- Together, these tools allow secure and flexible user account management on Linux systems

- **Expiration**
  - Expiration
    - Details of account expiration and password aging using configuration files and the chage command
  - *Configuration files*
    - Configuration files define system-wide account expiration and password aging rules
    - Used to enforce security policies requiring periodic password changes or account deactivation
    - */etc/login.defs*
      - Contains default password aging settings
      - PASS_MAX_DAYS sets maximum number of days a password is valid
      - PASS_MIN_DAYS sets minimum number of days before a password can be changed
      - PASS_WARN_AGE sets number of days before password expiration to warn user
      - Example to set max password age
        - PASS_MAX_DAYS90
    - */etc/passwd*

- Contains user-specific account information
- Stores expiration date in the last field as number of days since January 1, 1970
- Example view command
  - cat /etc/passwd
- Example line
  - alice:x:1001:1001:Alice User:/home/alice:/bin/bash:1680744000
- 1680744000 indicates the expiration date in epoch time
- Modifying this file directly is possible but not recommended

- *chage command*
  - chage manages password aging and account expiration for individual users
  - Used to override system-wide settings on a per-user basis
  - Allows setting of password expiration dates and account inactivity periods
  - Example command to set account expiration
    - sudo chage -E 2025-12-31 alice
  - Sets alice's account to expire on December 31, 2025
  - After expiration, login will be denied unless changed
- Summary
  - /etc/login.defs provides default settings applied to all accounts
  - /etc/passwd can store individual expiration dates but is rarely modified directly
  - chage provides flexible and safer control over individual account settings

- **User Information**
    - User Information Overview
        - Details of commands for retrieving and managing user information in Linux
    - *whoami command*
        - Displays the username of the currently logged-in user
        - Useful for confirming active session identity
        - Example command
            - whoami
        - Example output
            - admin
        - Returns the name of the user executing the command
    - *id command*
        - Displays detailed information about the current user
        - Includes user ID (UID), group ID (GID), and group memberships
        - Useful for managing file permissions and troubleshooting access issues
        - Example command
            - id
        - Example output
            - uid=1001(admin) gid=1001(admin) groups=1001(admin),27(sudo)
        - UID is the user's unique identifier
        - GID is the primary group's identifier
        - Groups lists all groups the user belongs to
    - *groups command*
        - Shows all groups the current user belongs to
        - Focuses on group names instead of numeric IDs

- Useful for verifying user access and permissions
- Example command
  - groups
- Example output
  - admin sudo
- Confirms group memberships for the current user

- *getent passwd command*
  - Queries the system's user database for all user entries
  - Retrieves data for both local and networked users
  - Useful for auditing and troubleshooting user-related issues
  - Example command
    - getent passwd
  - Example output
    - admin:x:1001:1001:Admin User:/home/admin:/bin/bash
    - alice:x:1002:1002:Alice User:/home/alice:/bin/bash
    - bob:x:1003:1003:Bob User:/home/bob:/bin/bash
  - Output includes username, UID, GID, home directory, and login shell
- Summary
  - whoami confirms session user identity
  - id reveals UID, GID, and all group memberships for permission management
  - groups identifies the group affiliations of the current user
  - getent passwd provides complete user account listings from local or networked sources

- **Login/Session Tracking**
  - Overview
    - Commands covered for login/session tracking
      - who
      - w
      - lastlog
      - last
  - *who*
    - Shows currently logged-in users
    - Displays username, terminal (TTY), login time, and source IP or hostname
    - Example output
      - alice pts/0 2025-04-24 10:23 (:0)
      - bob pts/1 2025-04-24 10:45 (192.168.1.10)
    - Provides quick snapshot of active user sessions
  - *w*
    - Gives detailed view of logged-in users and their activity
    - Displays system uptime, load average, user idle time, and running processes
    - Example output

```
10:50:24 up 2 days,  3:10,  2 users,  load average: 0.15, 0.10, 0.08
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
alice    pts/0    :0               10:23    5.00s  0.08s  0.00s -bash
bob      pts/1    192.168.1.10     10:45    0.00s  0.06s  0.01s sshd: bob [priv]
```

    - Useful for monitoring real-time user behavior and system performance
  - *lastlog*
    - Displays last login time for every user on the system
    - Helps identify inactive accounts or accounts that never logged in

- Example output

```
Username    Port     From             Latest
alice       :0       -                2025-04-24 10:23
bob         pts/1    192.168.1.10     2025-04-24 10:45
charlie     **never logged in**
```

- Great for auditing and cleanup of unused accounts

- *last*
  - Shows full login history for users, including reboots and shutdowns
  - Displays session start/end times and system events
  - Example output

```
bob      pts/1   192.168.1.10  Wed Apr 24 10:45   still logged in
alice    pts/0   :0            Wed Apr 24 10:23 - 10:45  (00:22)
reboot   system boot 5.4.0-66  Wed Apr 24 10:00   still running
```

  - Ideal for deep historical audits and security investigations

- Summary
  - who
    - Shows current users and when/where they logged in
  - w
    - Shows what current users are doing and how long they've been idle
  - lastlog
    - Shows each user's last login time, or if they never logged in
  - last
    - Displays a full log of previous sessions, reboots, and shutdowns

- **User Profile Templates**
  - User Profile Templates Overview
    - Details of user profile templates used to configure default and system-wide user environments in Linux
  - */etc/skel*
    - /etc/skel contains default files and directories copied to a new user's home directory upon account creation
    - Ensures consistency in file structure and environment settings for all new users
    - Administrators can place configuration files in /etc/skel
    - Common files include shell environment files and personal settings such as .bashrc
    - Example line in /etc/skel
      - alias ll='ls -l'
    - Sets a shortcut for the ls -l command
    - Files from /etc/skel are copied during user account creation to initialize the user environment
  - */etc/profile*
    - /etc/profile sets system-wide environment variables and configurations
    - Applied to all users who log in through a login shell
    - Useful for defining shared settings like PATH variables, limits, and shell preferences
    - Executed each time a user logs into the system
    - Example line in /etc/profile
      - export PATH=$PATH:/usr/local/bin
    - Adds /usr/local/bin to the system PATH for all users

        - Ensures users can execute programs from the specified directory
    - Summary
        - /etc/skel defines the initial environment and structure for new user accounts
        - /etc/profile applies global settings for all users upon login
        - Together, these files help administrators standardize and manage user environments across the system

- **Account Files**
    - Account Files
        - Details of essential files used for managing user accounts, group memberships, and password security in Linux
    - */etc/passwd*
        - Stores basic user account information
        - Contains fields for username, user ID (UID), group ID (GID), home directory, and default shell
        - Readable by all users
        - Does not store sensitive password data
        - Each line represents a user account and uses colon-separated fields
        - Example line
            - alice:x:1001:1001:Alice User:/home/alice:/bin/bash
                - x indicates that the password is stored in /etc/shadow
                - UID is 1001
                - GID is 1001
                - Home directory is /home/alice

- Default shell is /bin/bash

- */etc/group*
  - Manages user group information
  - Defines group name, group password (if any), group ID (GID), and list of group members
  - Used to control access to resources based on group membership
  - Each line corresponds to one group and is colon-separated
  - Example line
    - admin:x:1001:alice,bob
      - Group name is admin
      - GID is 1001
      - Members are alice and bob
      - x indicates no group password is set
- */etc/shadow*
  - Stores sensitive password information securely
  - Readable only by root
  - Contains hashed passwords and password aging details
  - Used for managing password policies, expiration, and account lockouts
  - Example line
    - alice:6saltsalt$3MvXZBpzErPr3wtygIlw1eF0DTKqG8ChkcIaqFjmcded8BHv36rrPzRgC2AmQXZo5B6kp5KNhFg2d4syEG9Sw0:18295:0:99999:7:::
      - Field 2 is the hashed password
      - Field 3 indicates days since password was last changed (18295)
      - Field 4 is minimum days between changes (0)

        - Field 5 is maximum days password is valid (99999)
        - Field 6 is days before expiration to warn user (7)
        - Final ::: are unused fields for account lockout and inactivity settings
    - Summary
        - /etc/passwd provides basic user information but omits sensitive data
        - /etc/group defines group structures and memberships
        - /etc/shadow protects password hashes and enforces aging policies

- **Attributes**
    - Overview
        - Linux tracks user and group identities using numeric attributes
        - Stored in /etc/passwd for users and /etc/group for groups
        - Used for file ownership, permissions, and access control
        - Four core attributes
            - UID
            - GID
            - EUID
            - EGID
    - *UID (User ID)*
        - Unique integer identifying each user account
        - Stored in /etc/passwd
        - Every file created by the user is stamped with this UID
        - Used during permission checks for ownership and access
        - UID alignment is important across systems for shared storage (e.g., NFS)

- *GID (Group ID)*
  - Identifies the user's primary group
  - Stored in /etc/group
  - Used to assign group permissions for shared access
  - Enables collaborative file access without duplicating data or modifying ACLs
  - Example
    - GID 2000 for group "developers" allows shared write access
- *EUID (Effective User ID)*
  - Temporary identity used when running set-uid (SUID) programs
  - Program runs with the UID of the file owner, not the user
  - Common example
    - /usr/bin/passwd runs with root's EUID
  - Allows privilege escalation for specific tasks
  - Kernel uses EUID for access checks during execution
  - Helps provide least-privilege access for trusted operations
- *EGID (Effective Group ID)*
  - Temporary group identity used when running set-gid (SGID) programs
  - Program or file runs with the group ID of the file
  - Files created in SGID directories inherit the directory's group
  - Example
    - chmod g+s <filename>
      - Sets SGID on a file
  - Enables temporary group access without permanently adding users to the group
  - Kernel uses EGID for checking group-based permissions

- Summary
  - UID and GID are stored in /etc/passwd and /etc/group
  - UID identifies the user
  - GID identifies the user's primary group
  - EUID and EGID provide temporary access during execution of set-uid/set-gid programs
  - Help maintain security and controlled access during privileged operations
  - Understanding these attributes aids in managing access and troubleshooting permission errors

- **Account Comparisons**
  - Account Comparisons
    - Understanding the three families of Linux accounts used in local account management
  - *User Accounts*
    - Created for people such as students, developers, and administrators
    - Used for login, launching shells, and storing personal files
    - UID assignment
      - Start at 1000 and increment upward on modern Linux distributions
      - First created user receives UID 1000
      - Subsequent users receive 1001, 1002, and so on
    - Easy to identify in the /etc/passwd file due to UID range
  - *System Accounts*
    - Reserved for the operating system's core services

- UID assignment
  - UID 0 is root
  - UID 1–99 for Debian-based systems
  - UID 1–199 for Red Hat-based systems
- No interactive login shell
- Examples
  - dbus
  - systemd-network
  - syslog
- Changes to these accounts can damage system functionality

- *Service Accounts*
  - Also called daemon accounts
  - Created during the installation of optional services
  - UID assignment
    - UID 100–999
  - Purpose
    - Each service runs with its own identity
    - Isolates processes for security and resource control
  - Login restrictions
    - Password field is locked
    - Login shell is set to /usr/sbin/nologin
  - Prevents interactive login while allowing background tasks
- Summary
  - The /etc/passwd file categorizes accounts by UID
  - User Accounts UID 1000 and above
  - System Accounts UID 0 and 1–99 or 1–199 depending on distro

- Service Accounts UID 100–999
- Helps ensure clean, secure, and isolated local account management

Processes

Objective 2.3: *Given a scenario, manage processes and jobs in a Linux environment*

- **Process ID**
  - Definition and Purpose
    - Linux kernel tags every running program with a unique PID
    - PID is used to allocate CPU time memory pages file-descriptor tables and scheduling priorities
    - Each process also has a PPID that identifies its parent process
    - PPID enables tracing of process lineage and understanding of process trees
  - *PID (Process ID)*
    - Unique non-repeating integer assigned to every running program
    - Allows administrators to send signals adjust priorities and monitor resource usage
    - PIDs can be discovered using
      - ps aux
      - ps -eo pid,stat,cmd
      - top
      - htop
    - Live kernel statistics can be accessed using
      - cat /proc/<PID>/
    - Signals can be sent using
      - kill -9 <PID>

        - SIGKILL is used to forcefully close a hung task
    - Priority can be changed using
        - renice -n 10 -p <PID>
    - System calls can be traced using
        - strace -p <PID>
    - PID 1 is systemd
        - First process started
        - Never exited from
        - Terminating PID 1 crashes the system
    - PIDs recycle after a process ends
        - Confirm current PID before sending commands
- *PPID (Parent Process ID)*
    - PPID records the process that spawned the current process
    - Creates a living process tree
    - Hierarchy can be viewed using
        - ps -e --forest -o pid,ppid,cmd
        - pstree -p
    - Stopping a parent process affects all its children
        - kill <PPID> sends signal to parent
        - pkill -P <PPID> <signal> targets all children
        - systemctl kill --kill-who=all <service>.service used in systemd environments
    - *Orphaned processes*
        - Are adopted by PID 1
    - *Zombie processes*
        - Are defunct processes not reaped by parent

- Reaping means parent performs a wait operation
- Frees child's resources and removes its entry from the process table

- Summary
  - Tracking PPIDs helps diagnose persistent background tasks
  - Verifies if forking daemons launched correctly
  - Enables cleanup of sub-trees without targeting individual PIDs
  - Helps maintain system stability through accurate process control

- **Process States**
  - Definition and Purpose
    - Every Linux process is tagged with a process state
    - Process state informs whether the task is active waiting paused or finished
    - Allows identification of performance issues and debugging needs
  - Process States in Linux
    - Running
    - Sleeping
    - Blocked
    - Stopped
    - Zombie
  - *Running State*
    - Marked with R in tools like top or ps
    - Process is either using a CPU or is first in line to get CPU access
    - Example

        - Cashier actively ringing up items
    - Helps identify where CPU time is being used
    - Single R process across all cores may need throttling or optimization
- *Sleeping State*
    - Most common state for processes
    - Annotated with S for interruptible sleep
        - Process can wake with signals like SIGINT
    - Annotated with D for uninterruptible sleep
        - Kernel waits for I/O completion like disk or network response
    - Knowing sleep type helps determine if a process can be nudged or needs further investigation
- *Blocked State*
    - Special uninterruptible sleep state
    - Indicates the process is waiting on slow resources
    - Annotated with D in top or ps
    - Example
        - Clerk at ticket counter waiting for jammed printer
    - Multiple D states can indicate issues with
        - Storage array
        - Kernel resource locks
        - Networked filesystems
- *Stopped State*
    - Process paused manually or by debugger
    - Annotated with T
    - Ctrl-Z moves a process into stopped state
    - Memory and file handles are preserved

        - Often used during debugging
        - Can be resumed using
            - kill -CONT <pid>
            - fg command
    - *Zombie State*
        - Process has completed but exit status is not collected by parent
        - Marked with Z or <defunct> in tools
        - Kernel holds the exit status until parent performs wait()
        - One or two zombies are harmless
        - Large numbers suggest parent is not cleaning up properly
        - Excessive zombies can cause process table overflows
        - Often reveals bugs in daemons or shell scripts
    - Summary
        - Running processes actively use CPU or wait for it
        - Sleeping processes await events with or without the ability to wake from signals
        - Blocked processes highlight slow I/O or resource contention
        - Stopped processes are paused and await resumption
        - Zombie processes have ended but await cleanup by parent

- **Process Priority**
    - Definition and Purpose
        - Process priority controls how aggressively a process competes for CPU time
        - Each process has a niceness value that influences scheduling

- Niceness range is from –20 (highest priority) to 19 (lowest priority)

- *nice Command*
  - Sets the initial niceness value when launching a process
  - Syntax
    - nice [-n adjustment] command [arguments…]
  - Example
    - nice -n 15 tar -czf /srv/backups/home-$(date +%F).tgz /home
  - Explanation
    - -n 15 sets niceness to 15
    - tar -czf creates a compressed archive
    - /srv/backups/home-$(date +%F).tgz is the output archive
    - /home is the directory to be backed up
  - Monitoring output using
    - ps -o pid,ni,cmd -C tar
  - Example output
    - 10234 15 tar -czf /srv/backups/home-2025-04-30.tgz /home
- *renice Command*
  - Modifies niceness value of an already running process
  - Requires super-user privileges to raise process priority
  - Syntax
    - renice [-n adjustment] { -p pid | -g pgid | -u user }
  - Example
    - sudo renice -n -5 -p 10234
  - Explanation
    - sudo grants super-user privileges
    - -n -5 sets new niceness value 5 points higher in priority

        - -p 10234 targets process ID 10234
    - Example output
        - 10234 old priority 15, new priority 10
- Summary
    - Niceness influences process scheduling priority
    - nice sets initial priority when starting a process
    - renice changes priority of an existing process
    - Higher priority is achieved by lowering niceness value
    - Super-user access required to increase priority
    - These tools offer dynamic control over system responsiveness and load

- **Process Monitoring**
    - Overview
        - Monitors active processes and system behavior
        - Toolkit includes ps, top, htop, and atop
    - *ps*
        - Provides static snapshot of running processes
        - ps -e
            - Lists every process
        - ps -a
            - Shows processes from other users terminals
        - ps -u
            - Username displays ownership CPU and memory usage
        - ps -f
            - Full format with parent PIDs and start times

        - ps -p <PID>
            - Focuses on a specific task
    - *top*
        - Live dashboard updating every few seconds
        - Shows load averages, uptime, CPU, and memory usage
        - Shift P
            - Sorts by highest CPU usage
        - Shift M
            - Sorts by memory usage
        - k
            - Kills a selected process
        - r
            - Renices a selected process
    - *htop*
        - Colorized interface with CPU RAM and swap bars
        - Scrollable list and tree view of parent child relationships
        - F6
            - Selects sort column
        - F3
            - Searches for a process name
        - F9
            - Sends signals such as TERM or KILL
    - *atop*
        - Displays live metrics for CPU memory disk and network per process
        - Logs snapshots at intervals to /var/log/atop for historical review
        - Allows replay to identify timing of resource spikes

- Summary
  - ps gives a static filtered view of processes
  - top offers a continually updating overview with interactive controls
  - htop enhances top with color navigation and hierarchical views
  - atop combines real time metrics with persistent logging to analyze past performance

- **Performance Metrics**
  - Overview of Performance Metrics
    - Performance metrics are tools used to monitor CPU and process activity
    - Linux systems use the sysstat package to collect, report, and log system performance data
    - Tools include mpstat and pidstat for system-level and process-level monitoring
  - *Sysstat Package*
    - Collection of utilities for performance monitoring
    - Captures real-time data
    - Preserves data for later analysis
    - Helps identify trends
    - Aids in troubleshooting and capacity planning
  - *mpstat Command*
    - Provides per-processor breakdown of CPU activity
    - Reports on time spent in user mode
    - Reports on time spent in system mode
    - Reports on time spent in I/O wait

- Reports on idle time
- mpstat Syntax
    - Format: mpstat [-P { cpu | ALL }] [interval [count]]
    - -P cpu targets a specific core
    - -P ALL targets all cores
    - interval defines time between samples
    - count sets number of samples to collect
    - mpstat Example
        - Command
            - mpstat -P ALL 2 5
    - Samples all CPUs every 2 seconds
    - Collects 5 samples total
    - Useful for spotting load imbalance across cores
    - Helps detect sudden spikes in I/O wait
    - Supports workload balancing and performance bottleneck analysis

- *pidstat Command*
    - Monitors resource usage by process
    - Reports on CPU usage
    - Reports on memory statistics
    - Reports on I/O activity
    - Reports on thread statistics
    - pidstat Syntax
        - Format
            - pidstat [-u] [-r] [-d] [-p pid,…] [interval [count]]
        - -u displays CPU usage
        - -r shows memory statistics

- -d reveals I/O activity
- -p pid specifies process IDs to monitor
- interval sets time between samples
- count sets number of samples to collect
- pidstat Example
  - Command
    - pidstat -u -r -d -p 1234 1 3
- Monitors process 1234 every second
- Collects 3 samples
- Displays CPU, RAM, and disk usage for the process
- Helps identify resource-heavy processes
- Enables performance tuning and process management

- Summary
  - sysstat package offers tools for real-time performance monitoring
  - mpstat provides CPU usage stats per processor
  - pidstat gives resource usage stats per process
  - Tools help identify system and process-level bottlenecks
  - Support informed troubleshooting and capacity planning

- **Diagnostic and Debugging Tools**
  - Overview of Diagnostic and Debugging Tools
    - Used to inspect and troubleshoot running processes in Linux
    - Provide real-time visibility into process behavior
    - Include /proc/<PID>, pstree, lsof, and strace
  - */proc/<PID> Directory*

- Reveals information about a specific process
- Displays command line
- Displays environment variables
- Displays memory usage
- Displays open file descriptors
- No additional software required
- Content updates live as the process runs
- Example
  - /proc/2345/status shows memory usage
- Example
  - /proc/2345/cmdline shows how the process was launched
- Useful for diagnosing resource issues

- *pstree Command*
  - Shows hierarchical diagram of processes
  - Displays parent and child relationships
  - Helps track process lineage
  - Used to investigate zombie processes
  - Option: pstree -p includes process IDs
  - Helps determine if a process should be restarted or terminated
  - Example
    - Use pstree to verify if worker processes were respawned correctly after a web app update
- *lsof Command*
  - Lists all open files and the processes using them
  - In Linux, files include sockets, pipes, and devices
  - Option

        - lsof -i shows all open network sockets
    - Filters available by protocol, port, and remote address
    - Example
        - sudo lsof -iTCP:443 -sTCP:LISTEN shows process listening on HTTPS
    - Example
        - lsof -i@192.168.1.100 lists connections to a host
    - Helps detect port conflicts
    - Identifies unauthorized daemons
    - Verifies firewall rules
    - Maps network endpoints to PIDs for further action
- *strace Command*
    - Hooks into a running process or starts one under trace
    - Logs all system calls made by the process
    - System calls include open(), read(), write(), and connect()
    - Displays results of each call
    - Example
        - strace -e open,connect -p 3456 traces access attempts
    - Used to find missing files
    - Used to detect misconfigured permissions
    - Used to diagnose subtle I/O errors
    - Offers detailed insight beyond standard logs
- Summary
    - /proc/<PID> provides live process information without extra tools
    - pstree visualizes parent–child relationships between processes
    - lsof lists open files and sockets, supports filtering and PID tracing

- strace records system calls and signals, ideal for low-level debugging
- Combined, these tools help diagnose, trace, and resolve process-related issues efficiently

- **Process Limits**
  - Overview of Process Limits
    - Process limits prevent individual programs from overconsuming system resources
    - Ensure system stability and responsiveness
    - Key process limits include CPU Time, Memory Usage, Stack Size, Number of Open File Descriptors, and Core File Size
  - *CPU Time*
    - Defines total seconds a process may run on the CPU
    - Kernel stops process when CPU time is exhausted
    - Prevents single tasks from monopolizing CPU
    - Warns administrator if a task exceeds CPU budget
    - Helps with capacity planning and resource balancing
  - *Memory Usage*
    - Sets upper limit on a process's virtual address space
    - Includes RAM and swap space
    - Swap space is disk area used as RAM overflow
    - Protects against memory leaks and runaway applications
    - Forces fast failure on memory allocation errors
    - Helps identify peak demands and right-size resources
  - *Stack Size*

- Controls memory reserved for a process's call stack
- Call stack stores return addresses, parameters, and local variables
- Supports tracking of active function calls
- Prevents memory corruption and crashes from deep recursion
- Avoids over-reservation of stack memory
- Ensures efficient and safe memory allocation per process

- *Number of Open File Descriptors*
  - A file descriptor is an integer referencing an open file, socket, or pipe
  - Sets a maximum number of descriptors a process can use simultaneously
  - High-concurrency applications like web servers may need thousands
  - Exceeding limit causes errors like "too many open files"
  - May lead to dropped connections or data loss
  - Monitoring helps detect leaks from unclosed descriptors
  - Enables optimized application resource management
- *Core File Size*
  - Controls whether and how large a crash memory dump file will be
  - Dumps are useful for post-crash debugging
  - Unrestricted dumps can quickly consume disk space
  - Limiting size balances diagnostic needs and storage usage
- Summary
  - CPU Time limits process CPU duration
  - Memory Usage defines total address space including swap
  - Stack Size reserves memory for call stacks
  - Number of Open File Descriptors limits open files, sockets, or pipes
  - Core File Size determines crash dump creation and size
  - These limits help maintain system health and aid in troubleshooting

- **Job Control Commands**
  - Overview
    - Allows suspending, backgrounding, and foregrounding of running processes
    - Useful for multitasking and terminal management
    - Key commands
      - Ctrl + Z
      - jobs
      - bg
      - fg
  - *Ctrl + Z*
    - Suspends the current foreground process
    - Returns control to the shell without terminating the process
    - Useful for pausing long-running or hanging commands
    - Example use
      - Pause ping google.com to run system diagnostics
  - *jobs*
    - Lists all suspended and backgrounded processes
    - Displays job numbers, status, and original command
    - +
      - Indicates the default job
    - -
      - Indicates the next job after the default
    - Example output

        - [1]+ Stopped ping google.com
        - [2]- Running tail -f /var/log/syslog &
    - *bg*
        - Resumes a suspended job in the background
        - Syntax
            - bg %<job number>
        - Example
            - bg %1 resumes job 1 in the background
        - Useful for freeing up the terminal while letting a task continue
    - *fg*
        - Brings a suspended or background job to the foreground
        - Syntax
            - fg %<job number>
        - Example
            - fg %2 brings job 2 to the foreground
        - Useful when a process needs user input or live monitoring
    - Summary
        - Ctrl + Z suspends a running process
        - jobs lists all suspended and backgrounded tasks
        - bg %<job number> resumes a task in the background
        - fg %<job number> resumes a task in the foreground
        - These tools give administrators efficient control over active terminal processes

- **Background Execution Commands**
  - Overview of Background Execution Commands
    - Used to run commands without blocking the shell
    - Enable multitasking and uninterrupted execution in Linux
    - Common commands include & and nohup
  - *& Command*
    - Sends a command to the background
    - Immediately returns shell control to the user
    - Useful for long-running tasks
    - Example
      - tar -czf logs.tar.gz /var/log &
    - Starts compression in the background
    - Displays job number
    - Allows continuation of other tasks in the same terminal
  - *nohup Command*
    - Stands for “no hang-up”
    - Prevents a command from terminating after logout or disconnection
    - Ignores the SIGHUP signal
    - Ensures command continues to run after SSH session ends
    - Example
      - nohup mysqldump -u root -p mydatabase > backup.sql &
    - Runs a MySQL backup in the background
    - Ensures backup continues even if the terminal is closed
  - Combined Usage of nohup and &
    - nohup ensures the process ignores hang-up signals
    - & sends the process to the background

        - Together they allow long processes to run uninterrupted and independently
        - Useful for remote administrative tasks like database backups or software builds
    - Summary
        - & runs commands in the background and frees the terminal
        - nohup keeps commands running after logout
        - nohup used with & supports uninterrupted background execution
        - These tools enhance task management and automation in Linux

- **Signals and Process Termination**
    - Definition and Purpose
        - Signals are short standardized messages sent to processes
        - Used to control behavior such as reloading configuration or terminating
        - Help in managing and terminating processes manually
    - *Signals*
        - Signals instruct programs to take specific actions
        - Common signals
            - *SIGHUP*
                - Signal number 1
                - Instructs service to reload configuration without stopping
            - *SIGTERM*
                - Signal number 15
                - Requests graceful shutdown
            - *SIGKILL*

        - Signal number 9
        - Immediately terminates process without cleanup
    - Example
        - kill -1 1234 sends SIGHUP to process ID 1234
- *Ctrl + C*
    - Sends SIGINT (interrupt signal) from keyboard
    - Used to stop a running command in terminal
    - Example
        - During a ping command, pressing Ctrl + C stops the process immediately
- *kill Command*
    - Sends specified signal to a process using its PID
    - Default signal is SIGTERM (15)
    - SIGKILL (9) used for forceful termination
    - Example
        - kill -9 5678 forcefully stops process with PID 5678
    - Typically no output if successful
- *killall Command*
    - Sends signals to processes by name instead of PID
    - Option -u targets processes by user
    - Example
        - killall -u alice bash terminates all bash processes owned by user alice
    - Default signal is SIGTERM
    - No terminal output on success
- *pkill Command*

        - Matches processes by pattern name user ID or terminal ID
        - Useful when exact process name or PID is unknown
        - Example
            - pkill -9 backup sends SIGKILL to all processes with "backup" in their name
        - Terminates processes immediately without cleanup
        - No terminal output on success
    - Summary
        - Signals control process behavior such as shutdown or reload
        - Ctrl + C sends SIGINT to interrupt a running command
        - SIGHUP reloads configuration
        - SIGTERM requests graceful termination
        - SIGKILL forces immediate termination
        - kill targets processes by PID
        - killall targets all processes by name or user
        - pkill targets processes using patterns or attributes

- **Shell and Process Invocation**
    - Definition and Purpose
        - Shell interaction involves starting controlling and ending sessions
        - Ctrl + D and exec help manage process invocation and shell behavior
        - Ctrl + D signals end of input and closes session
        - exec replaces the current process with another
    - *Ctrl + D*
        - Sends an end-of-file (EOF) signal to the shell

        - Closes the current shell session gracefully
        - No specific command needs to be typed
        - Example
            - Pressing Ctrl + D after finishing terminal tasks results in
                - logout
        - Indicates shell session has ended and returns to login prompt
    - *exec Command*
        - Replaces current shell or process with a new one
        - Syntax
            - exec [command] [arguments]
        - Does not return to original shell after execution
    - Examples of exec Usage
        - Replace shell with SSH session
            - exec ssh admin@server02
            - Replaces current session with an SSH login to server02 as admin
        - Redirect all shell output to a file
            - exec > session.log 2>&1
                - Redirects standard output to session.log
                - 2>&1 redirects standard error to same destination as output
                - Future commands write output and errors to session.log
                - Screen will no longer display outputs
    - Summary
        - Ctrl + D closes shell session or signals end of input
        - exec replaces current process or shell without returning
        - exec can redirect output to a file by modifying stdout and stderr

- Useful for seamless task transitions or logging session activity

- **Job Scheduling**
  - Overview
    - Automates program or script execution at scheduled times
    - Tools include
      - crontab
        - For repetitive tasks
      - at
        - For one-time tasks
      - anacron
        - For periodic tasks that tolerate system downtime
  - *crontab*
    - Schedules recurring tasks at specific times and intervals
    - Entries can be user-specific or system-wide
    - Format
      - * * * * * /path/to/command
        - Minute (0–59)
        - Hour (0–23)
        - Day of month (1–31)
        - Month (1–12)
        - Day of week (0–6, where 0 is Sunday)
    - Edit crontab with
      - crontab -e
    - Example entry to run script daily at 11:30 PM

- 30 23 * * * /home/admin/backup.sh

- *at*
  - Schedules one-time jobs at a specified future time
  - Does not repeat tasks
  - Syntax
    - at [time]
  - Example usage
    - at 3:00 AM tomorrow
    - Input
      - systemctl restart httpd
    - End input
      - Ctrl + D
    - Output
      - Confirms job and execution time
- *anacron*
  - Runs periodic jobs based on day intervals
  - Designed for systems that aren't always on
  - Jobs stored in
    - /etc/anacrontab
  - Format
    - Period delay job-identifier command
      - period
        - Days between runs (e.g., 1 = daily, 7 = weekly)
      - delay
        - Minutes to wait after boot
      - job-identifier

        - Unique task name
    - command
        - Script or program to execute
- Example entry
    - 7 10 weekly_cleanup /home/admin/cleanup.sh
        - Runs every 7 days with a 10-minute delay after boot
- Summary
    - crontab is used for repeated tasks scheduled with specific timing fields
    - at schedules single execution tasks at a future time
    - anacron runs periodic jobs even after missed schedules due to downtime
    - Together, these tools offer flexible automation options for managing Linux tasks reliably and efficiently

Software Configuration and Management

Objective 2.4: *Given a scenario, configure and manage software in a Linux environment*

- **Package Managers by Distributions**
    - Overview
        - Automate finding, downloading, installing, and updating software
        - Manage dependencies automatically
        - Avoid manual compiling and configuration
    - Distribution-Based Package Managers
        - Debian-based systems use
            - dpkg
            - apt
        - RHEL-based systems use
            - yum
            - dnf
        - OpenSUSE-based systems use
            - zypper
    - Debian-based Package Managers
        - *dpkg*
            - Low-level tool for installing .deb packages manually
            - Instal
                - sudo dpkg -i package.deb
            - Remove
                - sudo dpkg -r <package-name>

- *apt*
  - Higher-level tool that handles dependencies
  - Install
    - sudo apt install <package-name>
  - Remove
    - sudo apt remove <package-name>
  - Update package index
    - sudo apt update
  - Upgrade installed packages
    - sudo apt upgrade
- RPM Note for Debian
  - rpm is not native to Debian but useful across environments
    - Install
      - rpm -i
    - Upgrade
      - rpm -U
    - Freshen
      - rpm -F
    - Remove
      - rpm -e
    - Use
      - -h
        - For hash
      - -v
        - For verbose
- RHEL-based Package Managers

- *yum*
  - Legacy package manager for RHEL-based systems
  - Install
    - sudo yum install <package-name>
  - Remove
    - sudo yum remove <package-name>
  - Update system
    - sudo yum update
  - Check for updates
    - sudo yum check-update
- *dnf*
  - Modern replacement for yum
  - Install
    - sudo dnf install <package-name>
  - Remove
    - sudo dnf remove <package-name>
  - Group install
    - sudo dnf group install "Group Name"
  - List groups
    - sudo dnf group list
  - List installed packages
    - sudo dnf list installed
  - Upgrade system
    - sudo dnf upgrade
  - View history
    - sudo dnf history

- OpenSUSE-based Package Manager
  - *zypper*
    - Clean syntax and user-friendly output
    - Install
      - sudo zypper install <package-name>
    - Remove
      - sudo zypper remove <package-name>
    - Update system
      - sudo zypper update
    - Refresh repository index
      - sudo zypper refresh
- Summary
  - Package managers automate installation and update processes across Linux systems
  - Debian-based systems use dpkg and apt
  - RHEL-based systems use yum and dnf
  - OpenSUSE-based systems use zypper
  - Despite different syntax, all managers offer efficient and consistent software handling

- **System-Level Package Management**
  - Purpose of System-Level Package Management
    - Manage software packages at the system level
    - Ensure secure installation and compatibility
    - Handle installation dependencies and prevent conflicts

- *Repositories*
  - Storage locations for software packages
  - Maintained by distribution developers or trusted third parties
  - Accessed using update commands
    - sudo apt update (Debian-based systems)
    - sudo dnf check-update (RHEL-based systems)
  - Provide tested and secure packages
  - Updated regularly for security patches and bug fixes
  - Feature updates align with distribution release cycle
  - Administrators may add remove or prioritize repositories
  - Ensure correct software versions for security and enterprise needs
- *Package Dependencies*
  - Most software requires additional packages to function
  - Dependencies are automatically installed by package managers
  - Examples
    - sudo apt install apache2 installs Apache and dependencies like libapr1 and libaprutil1
    - sudo dnf install httpd installs Apache and dependencies like apr and apr-util
  - Simplify installation process and reduce errors
  - Important in minimal environments and troubleshooting scenarios
  - Administrators must monitor dependency issues when packages fail to install
- *Conflicts*
  - Occur when two packages cannot coexist due to incompatible dependencies or shared resources

        - Examples
            - Installing multiple versions of the same database server
        - Debian-based systems
            - apt stops installation and explains incompatible packages
        - RHEL-based systems
            - dnf halts installation and displays error messages
        - Administrators must resolve conflicts by
            - Removing or replacing packages
            - Adjusting repositories
            - Selecting alternative software
    - Summary
        - Repositories store and organize software packages for secure and efficient access
        - Package dependencies are supporting packages required by software
        - Conflicts arise from incompatible packages or resource usage
        - Administrators must manage all three areas to maintain system stability and reliability

- **Manual or Specialized Installations**
    - Purpose of Manual or Specialized Installations
        - Provide alternatives to system package managers for installing software
        - Allow manual installation from source or using language-specific tools
    - *Source Installation*
        - Used when software is not in repositories or needs custom build
        - Typical steps

- ./configure
- make
- sudo make install

- Example: Installing mynewapp from source
  - ./configure checks for required tools and libraries
    - Example output
      - checking for gcc... found /usr/bin/gcc
      - checking for required libraries... ok
  - make compiles the program
    - Example output
      - Compiling source files... done
  - sudo make install places files system-wide
    - Example output
      - Installing files to /usr/local/bin... done
- Provides full control over installation
- Requires attention to dependencies and compatibility

- *Language-Specific Package Managers*
  - Automate installation of language-specific software and dependencies
  - Tools include pip for Python cargo for Rust npm for Node.js
    - pip (Python)
      - Syntax
        - pip install <package-name>
      - Example
        - pip install requests
        - Example output
          - Successfully installed requests-2.31.0

- cargo (Rust)
    - Syntax
        - cargo install <package-name>
    - Example
        - cargo install exa
        - Example output
            - Compiling exa v0.10.1
            - Finished release [optimized] target(s) in 2m 15s
- npm (Node.js)
    - Syntax
        - npm install <package-name>
        - npm install -g <package-name> for global installation
    - Example
        - npm install -g http-server
        - Example output
            - + http-server@14.1.1
            - added 48 packages from 25 contributors

- Summary
    - Source installation offers manual control through configuration and compilation
    - Language-specific package managers simplify setup within development environments
    - Tools like pip cargo and npm manage downloads updates and dependencies

        - Useful for installing custom or environment-specific software

- **Repository Management**
    - Overview
        - Repositories provide secure, trusted software sources
        - Linux systems rely on repositories for reliable installations and updates
        - Packages are verified using GNU GPG signatures to prevent tampering
        - Repositories can be enabled or disabled to control software sources
        - Third-party repositories offer additional software not found in official repositories
    - *GNU GPG Signatures*
        - GNU stands for "GNU's Not Unix"
        - GPG stands for GNU Privacy Guard
        - GPG is used to encrypt and digitally sign data
        - GPG signatures verify package authenticity and integrity
        - The system checks signatures against trusted keys automatically
        - If a signature does not match, installation is halted with a warning
        - Example command to manually add a GPG key on Debian-based systems
            - wget -qO - https://example.com/repo.key | sudo apt-key add -
    - *Enabling and Disabling Repositories*
        - Enables control over which sources are active
        - Disabling a repo prevents updates from that source
        - Temporary enabling can allow one-time installations
        - On Debian-based systems
            - Managed via /etc/apt/sources.list or /etc/apt/sources.list.d/

- Lines can be commented or uncommented to disable or enable

- On RHEL-based systems using dnf
  - Disable a repo
    - sudo dnf config-manager --set-disabled <repo-name>
  - Enable a repo
    - sudo dnf config-manager --set-enabled <repo-name>

- *Third-Party Repositories*
  - Maintained by individuals, companies, or independent projects
  - Not officially supported by Linux distribution maintainers
  - Useful for accessing software not available in official repos
  - Require careful evaluation due to security and quality concerns
  - Example
    - Docker repository is commonly added to install Docker with latest updates

- Summary
  - Repositories provide secure software and simplify updates
  - GNU GPG signatures verify the authenticity and integrity of packages
  - Administrators can enable or disable repositories to manage sources
  - Third-party repositories extend functionality but must be trusted

- **Debian-based Package and Repository Exclusions**
  - Purpose of Package and Repository Exclusions
    - Prevent specific packages from being updated automatically
    - Maintain system stability and avoid compatibility issues
    - Useful for production environments and critical software

- *apt-mark hold*
  - Locks a package at its current version
  - Prevents automatic upgrades during apt upgrade
  - Syntax
    - sudo apt-mark hold <package>
  - Example
    - sudo apt-mark hold nginx
  - Ensures the package is skipped during system updates
- *apt-mark showhold*
  - Displays list of held packages
  - Useful for auditing excluded packages
  - Syntax
    - apt-mark showhold <package>
  - Shows which packages are protected from upgrades
- *apt-mark unhold*
  - Removes hold on a package
  - Allows package to be updated during apt upgrade
  - Syntax
    - sudo apt-mark unhold <package>
  - Example
    - sudo apt-mark unhold nginx
  - Useful when stability or compatibility concerns are resolved
- *dpkg --set-selections*
  - Used for advanced or bulk package management
  - Sets package states programmatically
  - Common in automation scripts and deployments

        - Syntax
            - echo "<package> hold" | sudo dpkg --set-selections
        - Example
            - echo "nginx hold" | sudo dpkg --set-selections
        - Mirrors functionality of apt-mark hold with more flexibility
    - Summary
        - apt-mark hold prevents updates to specified packages
        - apt-mark showhold lists held packages
        - apt-mark unhold removes update restrictions
        - dpkg --set-selections manages package holds in bulk or scripted environments
        - These tools support stable and controlled updates on Debian-based systems

- **RHEL-based Package and Repository Exclusions**
    - Purpose of Package and Repository Exclusions
        - Prevent specific updates to maintain system stability
        - Limit which repositories are active and which packages update
        - Ensure consistent and controlled software environments
    - *yum/dnf-config-manager*
        - Used to enable or disable software repositories
        - Provides control over where packages are sourced
        - Syntax to disable a repository
            - sudo dnf config-manager --set-disabled <repository-name>
        - Syntax to enable a repository

        - sudo dnf config-manager --set-enabled <repository-name>
    - Useful for
        - Disabling unstable or development repositories
        - Enabling optional sources for specific packages
        - Enforcing stability security and compliance standards
- *dnf/yum versionlock*
    - Prevents specific packages from being upgraded
    - Helps maintain fixed versions for critical applications
    - Syntax to add version lock using dnf
        - sudo dnf versionlock add <package-name>
    - Syntax to delete version lock using dnf
        - sudo dnf versionlock delete <package-name>
    - Works similarly with yum on older systems
    - Plugin must be installed on systems using dnf or yum
    - Protects software from unexpected changes during updates
- Summary
    - yum-config-manager or dnf config-manager controls repository activation
    - dnf versionlock or yum versionlock prevents package upgrades
    - These tools help enforce system stability and predictable software behavior

- **openSUSE-based Package and Repository Exclusions**
    - Purpose of Package and Repository Exclusions
        - Maintain system stability and reliability by restricting updates
        - Prevent unauthorized or unstable changes to critical software

        - Control package updates and repository usage
    - *zypper al (add lock)*
        - Locks specific packages to prevent updates or removal
        - Syntax
            - sudo zypper al <package-name>
        - Example
            - sudo zypper al nginx
        - Prevents nginx from being updated during system upgrades
        - Useful for preserving stable versions in production environments
    - *zypper ll (list locks)*
        - Displays all currently locked packages
        - Syntax
            - sudo zypper ll
        - Helps administrators audit and confirm update restrictions
        - Important for tracking exclusions during system reviews
    - *zypper rl (remove lock)*
        - Removes a lock from a previously locked package
        - Syntax
            - sudo zypper rl <package-name>
        - Example
            - sudo zypper rl nginx
        - Allows package to be updated during the next upgrade cycle
        - Useful when updates are verified or required for security
    - *zypper mr -d (modify repository - disable)*
        - Disables a specific software repository
        - Syntax

        - sudo zypper mr -d <repo-name>
    - Prevents the disabled repository from being used for installs or updates
    - Useful for avoiding unstable sources or temporarily disabling unwanted updates
  - Summary
    - zypper al locks packages from being updated or removed
    - zypper ll lists all currently locked packages
    - zypper rl removes package locks when updates are needed
    - zypper mr -d disables repositories to control software sources
    - These tools support stable and consistent openSUSE system environments

- **Update Alternatives**
  - Purpose of Update Alternatives
    - Manage default programs when multiple versions are installed
    - Control which version of a tool is used by the system
    - Common for managing versions of Java Python editors and interpreters
  - *update-alternatives --list*
    - Displays all available installed options for a given tool
    - Syntax
      - sudo update-alternatives --list <tool-name>
    - Example
      - sudo update-alternatives --list java
    - Output shows paths to all known Java versions
    - Helps administrators understand what choices exist

- *update-alternatives --config*
  - Allows selection of the default version from available options
  - Syntax
    - sudo update-alternatives --config <tool-name>
  - Example
    - sudo update-alternatives --config java
  - Displays a numbered list of versions
  - Administrator selects the desired version by number
  - Avoids manual modification of system configuration files
- *update-alternatives --install*
  - Adds a new version as an alternative option
  - Syntax
    - sudo update-alternatives --install <generic_link> <name> <target> <priority>
  - Example
    - sudo update-alternatives --install /usr/bin/java java /opt/java/bin/java 100
  - /usr/bin/java is the generic link used in commands
  - java is the name of the alternative group
  - /opt/java/bin/java is the actual path to the program
  - 100 is the priority value used for automatic selection
  - Higher numbers have higher priority in default selection
- Summary
  - update-alternatives --list shows available program versions
  - update-alternatives --config sets the default program version
  - update-alternatives --install adds a new version to the alternatives system

- Provides flexible control over which versions are used by default in Linux systems

- **Software Configuration**
  - Purpose of Software Configuration
    - Define and control software behavior
    - Apply changes either system-wide or at the user level
    - Customize software functionality using environmental variables
  - System-Wide vs User-Level Configuration
    - *System-wide configuration*
      - Affects all users
      - Managed by administrators
      - Set in directories and files such as
        - /etc
        - /etc/sysctl.conf
      - Used for settings such as
        - Default shells
        - Service control
        - Resource limits
    - *User-level configuration*
      - Affects only the individual user account
      - Defined in hidden files in the user's home directory
        - .bashrc
        - .profile
      - Helps preserve consistency while allowing user flexibility

- *Environmental Variables*
  - Named values used to configure software behavior
  - Examples
    - PATH
    - HOME
    - EDITOR
  - System-wide environmental variables set in
    - /etc/environment
    - /etc/profile
  - User-specific environmental variables set in
    - .bashrc
    - .bash_profile
  - Administrators can extend PATH to include shared script directories
  - Users can set EDITOR to define a preferred text editor
  - Used to manage execution environments for users and processes
- Summary
  - System-wide settings apply to all users and are managed in /etc
  - User-level settings apply to individuals and are managed in home directory files
  - Environmental variables customize system and software behavior
  - These configurations enable flexible and controlled administration of Linux environments

- **Sandboxed Applications**
  - Purpose of Sandboxed Applications

- Run software in isolated environments
- Keep files settings and dependencies separate from the system
- Improve safety flexibility and ease of management

- *Snaps*
  - Developed by Canonical
  - Used widely on Ubuntu and Debian-based systems
  - Syntax to install
    - sudo snap install <package-name>
  - Example
    - sudo snap install vlc
  - View package info
    - snap info <package-name>
  - List installed snaps
    - snap list
  - Remove snap package
    - sudo snap remove <package-name>
  - Includes all dependencies to reduce version conflicts
  - Applications can be launched by typing their names
- *Flatpak*
  - Offers flexibility in choosing software sources
  - Repositories are called remotes
  - Syntax to install
    - flatpak install <remote-name> <application-id>
  - Example
    - sudo flatpak install flathub org.gimp.GIMP
  - Run installed applications

- flatpak run <application-id>
- Allows control over source selection
- Useful for testing new versions or using alternative remotes

- *AppImage*
  - Packages entire application in a single portable file
  - No installation required
  - No root access required
  - Steps to use
    - Download file
    - Mark executable
      - chmod +x <filename>.AppImage
    - Run application
      - ./<filename>.AppImage
  - Example
    - wget https://download.kde.org/stable/krita/5.2.0/krita-5.2.0-x86_64.appimage
    - chmod +x krita-5.2.0-x86_64.appimage
    - ./krita-5.2.0-x86_64.appimage
  - Does not integrate with native package management
  - Manual updates required
- Summary
  - Snaps provide centralized installation and auto updates
  - Flatpak offers isolation with flexible repository options
  - AppImage enables portable usage without installation
  - All three tools help manage secure and isolated software environments

- **Core-level Service Configuration**
    - Purpose of Core-Level Services
        - Provide essential networking and timing functions
        - Support communication resource resolution and time synchronization
    - *DHCP (Dynamic Host Configuration Protocol)*
        - Automatically assigns IP addresses and network settings
        - Scalable alternative to static IP configuration
        - Server configuration file
            - /etc/dhcp/dhcpd.conf
        - Client configuration file
            - /etc/dhclient.conf
        - Configurable options
            - domain-name
            - domain-name-servers
            - routers
            - ntp-servers
            - broadcast-address
            - host-name
            - netmask
        - Ports
            - UDP port 67 (server)
            - UDP port 68 (client)
        - IP lease process
            - DORA

        - Discover
        - Offer
        - Request
        - Acknowledge
    - Manual client invocation
        - sudo dhclient eth0
    - Reduces administrative effort and prevents IP conflicts
- *DNS (Domain Name System)*
    - Resolves domain names into IP addresses
    - Local configuration file
        - /etc/resolv.conf
    - Uses
        - UDP port 53 (queries)
        - TCP port 53 (zone transfers large queries)
    - Manual mappings file
        - /etc/hosts
    - Resolution order file
        - /etc/nsswitch.conf
    - Important for
        - Web browsing
        - Email
        - Package management
    - Enables troubleshooting for slow lookups and spoofing risks
- *NTP (Network Time Protocol)*
    - Synchronizes system clocks
    - Operates over UDP port 123

- Traditional configuration file
  - /etc/ntp.conf
- Modern alternative Chrony
  - Configured in /etc/chrony.conf
- Features
  - Fast synchronization
  - Drift tracking
  - Stratum hierarchy
    - Stratum 0: Reference (atomic GPS)
    - Stratum 1: Directly linked to Stratum 0
    - Stratum 2: Linked to Stratum 1

- *PTP (Precision Time Protocol)*
  - Provides sub-microsecond time synchronization
  - Used in environments needing high accuracy
    - Industrial automation
    - Scientific instrumentation
    - Financial systems
  - Configured with
    - ptp4l
    - Settings in /etc/linuxptp/
  - Requires hardware timestamping support
- Summary
  - DHCP automates IP address configuration via UDP ports 67 and 68 using the DORA sequence
  - DNS resolves hostnames using a layered structure of resolv.conf hosts and nsswitch.conf

- NTP synchronizes time via UDP port 123 using stratums or Chrony
- PTP offers precise time coordination using ptp4l and hardware timestamping

- **Application-level Service Configuration**
  - Overview
    - Application-level services manage communication and access between users and systems
    - Each service has a specific network-based role
    - Services are configured individually based on their functions
    - Common services include
      - HTTP
      - SMTP
      - IMAP4
  - *HTTP – Web Content Delivery*
    - HTTP stands for Hyper Text Transfer Protocol
    - Delivers websites and web applications
    - Uses TCP port 80 for HTTP and port 443 for HTTPS
    - Two common tools
      - Apache (httpd)
      - Nginx
    - *Apache Configuration*
      - Main config file
        - /etc/httpd/conf/httpd.conf
      - Additional configs

        - /etc/httpd/conf.d/
    - Configurable elements
        - Server directives
        - Document root paths
        - Virtual hosts
        - Access rules
        - Logging formats
    - Systemd commands
        - Start
            - sudo systemctl start httpd
        - Enable on boot
            - sudo systemctl enable httpd
        - Status check
            - sudo systemctl status httpd
- *Nginx Configuration*
    - Stands for Engine-X
    - Lightweight, efficient, scalable
    - Used as a web server and reverse proxy
    - Main config file
        - /etc/nginx/nginx.conf
    - Site configs
        - /etc/nginx/conf.d/ or /etc/nginx/sites-available/
    - Config sections include
        - Server behavior
        - Location directives
        - SSL settings

        - Proxy rules
            - Can serve as a front-end for Apache or application servers
    - *SMTP – Email Sending*
        - SMTP stands for Simple Mail Transfer Protocol
        - Sends email between servers and clients to servers
        - Default port
            - TCP 25
        - Secure/authenticated submission ports
            - Port 587 (submission)
            - Port 465 (SMTPS)
    - *IMAP4 – Email Retrieval and Organization*
        - IMAP4 stands for Internet Message Access Protocol version 4
        - Allows retrieval and organization of email stored on a server
        - Supports multi-device synchronization
        - Does not delete email after retrieval like POP3
        - Default port
            - TCP 143
        - Secure port
            - TCP 993 (IMAPS)
    - Summary
        - HTTP delivers websites using Apache or Nginx
        - SMTP sends emails using ports 25, 587, and 465
        - IMAP4 retrieves and organizes email using ports 143 and 993
        - Each service is configured through specific files and ports

Systemd

Objective 2.5: *Given a scenario, manage Linux using systemd*

- **Units**
  - Overview of systemd Units
    - *systemd*
      - Is responsible for managing the system startup and service behavior
    - Unit files define the configuration for specific system components
    - Unit types include
      - Services
      - Targets
      - Mounts
      - Timers
  - *Services – Managing Daemons and Applications*
    - Service units manage background programs and daemons
    - Files use the .service extension
    - Service unit file sections
    - [Unit]
      - Description=
        - Provides summary
      - Requires= and Wants=
        - Define dependencies
      - Before= and After=

        - Define startup order
- [Service]
    - Type=
        - Defines process behavior
        - simple for foreground scripts
        - forking for daemons
    - ExecStart=
        - Specifies the start command
    - ExecStop=
        - Defines the stop method
    - User=
        - Defines non-root execution account
- [Install]
    - WantedBy= or RequiredBy=
        - Defines startup target
    - Used with systemctl enable to create symbolic links
- Service Unit Example

```
[Unit]
Description=Start custom app after network is up
After=network.target

[Service]
Type=simple
User=adminuser
ExecStart=/usr/local/bin/app.sh
ExecStop=/bin/kill $MAINPID

[Install]
WantedBy=multi-user.target
```

    - Description=
        - Explains the unit

- After=network.target
  - Delays startup until networking is ready
- Type=simple
  - Indicates non-daemon behavior
- User=adminuser
  - Runs service under a non-root account
- ExecStart=
  - Specifies full path to script to be executed
- ExecStop=
  - Defines how to terminate process using kill and $MAINPID to reference script's main process
- WantedBy=multi-user.target
  - Enables start at normal system boot
- Managed with
  - sudo systemctl start app
  - sudo systemctl enable app
  - sudo systemctl status app

- *Targets – Defining System States*
  - Targets group services to define system states
  - Types of targets
    - *multi-user.target*
      - Non-graphical mode with networking
    - *graphical.target*
      - GUI mode including multi-user.target
    - *network-online.target*
      - Ensures full network connectivity

- Target Management
  - View current default target
    - systemctl get-default
  - Set new default target
    - systemctl set-default graphical.target
  - Used with WantedBy= in [Install] sections

- *Mounts – Automating File System Mounts*
  - Mount units use .mount files to define file system mounts
  - Preferred over /etc/fstab for systemd integration
  - Sections include [Unit], [Mount], and [Install]
  - Mount Unit Example

```
[Unit]
Description=Mount external USB drive
After=local-fs.target

[Mount]
What=/dev/sdb1
Where=/mnt/usbdrive
Type=ext4
Options=defaults

[Install]
WantedBy=multi-user.target
```

    - Description=
      - Explains the mount purpose
    - After=local-fs.target
      - Ensures proper timing
    - What=
      - Identifies device or source
    - Where=
      - Sets mount location
    - Type=

- Sets file system type
- Options=
  - Controls behavior (defaults, ro, noatime)
- WantedBy=multi-user.target
  - Ensures mounting during boot
- Enable and start mount
  - sudo systemctl enable mnt-usbdrive.mount
  - sudo systemctl start mnt-usbdrive.mount

- *Timers – Scheduling Tasks*
  - Timer units use .timer files
  - Replace or supplement cron jobs
  - Paired with .service files
  - Directives
    - OnBootSec=
      - Defines delay after boot
    - OnCalendar=
      - Uses calendar syntax
    - Persistent=true
      - Ensures missed runs happen on boot
  - Timer and Service Example
    - backup.service

```
[Unit]
Description=Run backup script

[Service]
Type=oneshot
ExecStart=/usr/local/bin/backup.sh
```

    - backup.timer

```
[Unit]
Description=Run backup daily at 2AM

[Timer]
OnCalendar=*-*-* 02:00:00
Persistent=true

[Install]
WantedBy=timers.target
```

        - Type=oneshot
            - Runs once and exits
        - OnCalendar=*-*-* 02:00:00
            - Triggers at 2:00 AM daily
        - Persistent=true
            - Ensures it runs even if missed during downtime
        - Enable with systemctl enable for boot startup
    - Summary
        - Services manage background programs and include multiple configuration sections
        - Targets define system states and group units for startup behavior
        - Mounts provide reliable, timed file system mounting
        - Timers schedule recurring tasks with advanced control and recovery
        - All unit types work together to streamline Linux system management

- **Management Utilities**
    - Purpose of Management Utilities
        - Control system services and behavior
        - Set system identity

        - Modify kernel-level configuration parameters
    - *systemctl*
        - Used to control services managed by systemd
        - Start a service
            - sudo systemctl start <servicename>
        - Stop a service
            - sudo systemctl stop <servicename>
        - Restart a service
            - sudo systemctl restart <servicename>
        - Enable service to start at boot
            - sudo systemctl enable <servicename>
        - Disable service from starting at boot
            - sudo systemctl disable <servicename>
        - View service status
            - sudo systemctl status <servicename>
        - Mask a service to block all activation
            - sudo systemctl mask <servicename>
        - Essential for stable and predictable service management
    - *hostnamectl*
        - Used to manage system hostname and identity
        - Set a permanent hostname
            - sudo hostnamectl set-hostname <hostname>
        - Example
            - sudo hostnamectl set-hostname webserver1
        - Useful in enterprise and remote SSH environments
    - *sysctl*

- Used to view and modify Linux kernel parameters at runtime
- View all parameters
  - sysctl -a
- Temporarily change a parameter
  - sudo sysctl -w key=value
- Load parameters from default configuration
  - sudo sysctl -p
- Apply changes from a specific configuration file
  - sudo sysctl -p /path/to/file
- Apply recursively to all subkeys under a base key
  - sysctl -r
- Ignore errors when applying parameters
  - sysctl -e
- Persistent changes configured in
  - /etc/sysctl.conf
  - /etc/sysctl.d/*.conf
- Example for enabling IP forwarding
  - net.ipv4.ip_forward = 1
  - Add to /etc/sysctl.conf and run sudo sysctl -p

- Summary
  - systemctl manages services under systemd
  - hostnamectl sets or views system hostname
  - sysctl adjusts runtime kernel parameters and ensures persistent system tuning
  - These utilities are core tools for reliable Linux system administratio

- **Configuration Utilities**
    - Overview
        - Configuration utilities in Linux include tools for managing DNS and time settings
        - Key utilities are systemd-resolved, resolvectl, and timedatectl
    - *systemd-resolved*
        - systemd-resolved is a system service for managing DNS name resolution
        - Handles DNS servers, search domains, and DNS caching
        - Works with NetworkManager, Netplan, and systemd-networkd
        - Communicates via /etc/resolv.conf, often linked to a stub resolver
        - Stub resolver receives DNS queries and forwards to upstream DNS servers
        - Centrally manages DNS query processing and routing
    - *resolvectl*
        - resolvectl is a command-line tool for systemd-resolved
        - Replaces older tools like dig, host, and nslookup
        - Displays DNS configuration
            - resolvectl status
                - Shows DNS servers, interfaces, and scopes
        - Queries DNS resolution
            - resolvectl query <hostname>
                - Returns IP address and DNSSEC status
        - Views interface-specific DNS servers
            - resolvectl dns <interface>
        - Shows which interface is used for default DNS routing
            - resolvectl default-route
    - *timedatectl*

        - timedatectl is a utility for viewing and managing time settings
        - Checks system clock and NTP status
            - timedatectl status
        - Changes time zone
            - sudo timedatectl set-timezone <Region/City>
        - Example
            - sudo timedatectl set-timezone America/New_York
        - Sets system time manually
            - sudo timedatectl set-time "YYYY-MM-DD HH:MM:SS"
        - Enables or disables NTP synchronization
            - sudo timedatectl set-ntp <true or false>
    - Summary
        - systemd-resolved manages DNS resolution through a stub resolver
        - resolvectl provides inspection and testing of DNS settings
        - timedatectl manages time zones, clocks, and synchronization
        - Together, these utilities ensure accurate time and reliable DNS on Linux systems

- **Analysis and Diagnostic Utilities**
    - Purpose of Analysis and Diagnostic Utilities
        - Assess system performance
        - Identify sources of slow boot times
        - Support performance optimization and troubleshooting
    - *systemd-analyze*
        - Summarizes total system boot time

- Breaks boot process into key stages
  - Firmware initialization
  - Kernel phase
  - User space
- Syntax
  - systemd-analyze
- Example output
  - Startup finished in 1.763s (firmware) + 2.113s (kernel) + 9.224s (userspace) = 13.100s
  - graphical.target reached after 9.187s in userspace
- Helps determine if delays are hardware, kernel, or service related

- *systemd-analyze blame*
  - Lists services by how long they take to start
  - Displays results in descending order by startup time
  - Syntax
    - systemd-analyze blame
  - Example output
    - 3.814s NetworkManager.service
    - 2.721s accounts-daemon.service
    - 1.843s systemd-logind.service
    - 1.202s snapd.service
  - Helps pinpoint slow or misconfigured services
  - Enables optimization through service investigation and comparison
- Summary
  - systemd-analyze provides boot time breakdown by stage

        - systemd-analyze blame identifies services with the longest startup durations
        - These tools help administrators troubleshoot delays and improve boot efficiency

- **Configuration and Reloading Unit States**
    - Purpose of Configuration and Reloading Unit States
        - Customize service behavior in systemd
        - Safely modify settings without changing original unit files
        - Ensure changes are applied correctly
    - *systemctl edit*
        - Creates or modifies a drop-in override file for a systemd unit
        - Does not modify the original unit file
        - Syntax
            - systemctl edit <unit>
        - Opens an editor to enter new or overridden settings
        - Example
            - sudo systemctl edit apache2.service
        - Example configuration
            - [Service]
            - TimeoutStartSec=60s
        - File saved at
            - /etc/systemd/system/apache2.service.d/override.conf
        - Allows safe and persistent customization of unit behavior
    - *systemctl daemon-reload*

- Reloads systemd's internal configuration cache
- Required after editing unit or override files
- Applies updated settings across all services
- Syntax
  - sudo systemctl daemon-reload
- No output upon success
- Must be followed by service restart to activate changes
- Example
  - sudo systemctl daemon-reload
  - sudo systemctl restart apache2.service

- Summary
  - systemctl edit safely customizes services using drop-in override files
  - systemctl daemon-reload reloads systemd's memory to recognize changes
  - Restarting the service after reload ensures settings take effect
  - Omitting daemon-reload prevents edits from being applied

- **Boot-Time State Management**
  - Overview
    - Boot-time state management controls when services start during Linux boot
    - systemd provides commands to enable, disable, mask, and unmask services
  - *enable*
    - enable configures a service to start automatically at boot

- Creates a symbolic link between the service unit file and a target
- Target examples include multi-user.target
- Syntax
  - sudo systemctl enable <service-name>
- Example
  - sudo systemctl enable apache2.service
- Output
  - Created symlink /etc/systemd/system/multi-user.target.wants/apache2.service → /lib/systemd/system/apache2.service
    - Links Apache to multi-user.target so it starts automatically at boot

- *disable*
  - disable prevents a service from starting automatically at boot
  - Removes the symbolic link created by enable
  - Does not stop a running service or block manual starts
  - Syntax
    - sudo systemctl disable <service-name>
  - Example
    - sudo systemctl disable apache2.service
  - Output
    - Removed /etc/systemd/system/multi-user.target.wants/apache2.service
      - Apache will not start at boot but can be started manually with sudo systemctl start apache2.service
- *mask*

- mask blocks a service from being started manually or automatically
- Replaces the service unit file with a symlink to /dev/null
- Syntax
  - sudo systemctl mask <service-name>
- Example
  - sudo systemctl mask apache2.service
- Output
  - Created symlink /etc/systemd/system/apache2.service → /dev/null
    - Any attempt to start Apache will fail

- *unmask*
  - unmask removes the mask from a service
  - Deletes the symlink to /dev/null
  - Allows the service to be enabled or started again
  - Syntax
    - sudo systemctl unmask <service-name>
  - Example
    - sudo systemctl unmask apache2.service
      - No output unless an error occurs
- Summary
  - enable starts services at boot by linking to system targets
  - disable removes this link but allows manual starting
  - mask prevents all forms of starting a service
  - unmask restores the ability to manage the service normally
  - These commands allow fine control of service startup behavior in Linux

- **Service Control**
    - Purpose of Service Control
        - Manage and monitor background services
        - Start stop restart or reload services using systemctl
        - Check service state for troubleshooting and performance
    - *status*
        - Displays current state and service details
        - Syntax
            - systemctl status <service-name>
        - Example
            - systemctl status cron.service
        - Output includes
            - Service description and loaded status
            - Active state and uptime
            - Main PID and memory usage
            - CGroup and process tree
        - Used to identify running inactive or failed services
    - *start*
        - Starts a service immediately
        - Does not enable service at boot
        - Syntax
            - sudo systemctl start <service-name>
        - Example
            - sudo systemctl start apache2.service
        - Follow-up command to confirm
            - systemctl status apache2.service

- *restart*
  - Stops and starts a service in one command
  - Used after configuration changes or to clear errors
  - Syntax
    - sudo systemctl restart <service-name>
  - Example
    - sudo systemctl restart ufw.service
  - No output if successful
  - status command confirms results
- *reload*
  - Reloads service configuration without stopping the service
  - Works only if service supports dynamic reloads
  - Syntax
    - sudo systemctl reload <service-name>
  - Example
    - sudo systemctl reload postfix.service
  - If unsupported systemd returns an error
  - Alternative to restart for live updates
- Summary
  - status checks the operational state of a service
  - start launches a service manually
  - restart refreshes a service by stopping and starting it
  - reload applies configuration changes without interruption
  - All commands are critical for managing Linux services effectively

Virtualization

Objective 1.7: *Summarize virtualization on Linux systems*

- **Bare metal vs virtual machines**
    - Overview of Bare Metal vs Virtual Machines
        - *Bare metal*
            - Refers to installing an operating system directly on physical hardware
        - *Virtual machines (VMs)*
            - Run on top of a hypervisor that shares hardware resources
        - Key components to compare
            - Physical Layer
            - Guest Operating Systems
    - Physical Layer – Hardware Access and Management
        - Bare metal
            - Operating system installs directly on hardware
            - Full exclusive access to CPU, memory, disks, and network interfaces
            - Ideal for high-performance environments
            - Common uses include
                - Database servers
                - Dedicated firewall appliances
        - Virtual machines
            - Hardware is abstracted by a hypervisor

- Hypervisors include
  - KVM
  - VMware ESXi
  - Hyper-V
- Hardware is shared among multiple VMs
- Each VM receives a virtual portion of host CPU, RAM, and disk
- Increases hardware utilization
- Reduces costs in enterprise environments

- Guest Operating Systems – Running Environments
  - Guest OS can be any Linux or other operating system
  - Functions like a full system
    - Own file system
    - Own services
    - Own users
    - Own network configuration
  - Bare metal
    - Guest OS accesses hardware directly
    - Delivers better performance
    - Ensures predictable system behavior
  - Virtual machines
    - Guest OS accesses virtualized hardware
      - Virtual network adapters
      - Virtual disks
    - May introduce performance overhead
    - Requires careful tuning and resource management
  - Admin considerations in VMs

        - Monitor shared resource constraints
        - Address competition among VMs
    - Summary
        - Bare metal installs OS directly on hardware for exclusive and optimized resource use
        - Virtual machines use a hypervisor to share physical hardware among multiple guest OSes
        - Guest OSes behave similarly but differ in hardware access
        - Bare metal delivers performance and predictability
        - Virtualization offers flexibility and efficient resource use with potential trade-offs

- **Hypervisors**
    - Purpose of Hypervisors
        - Enable virtualization on Linux systems
        - Allow creation and management of virtual machines
    - *KVM (Kernel-based Virtual Machine)*
        - Built-in feature of the Linux kernel
        - Allows Linux to act as a Type 1 hypervisor
        - Provides CPU and memory virtualization
        - Integrated into the kernel for high efficiency
        - Does not create or manage virtual machines on its own
        - Requires user-space tools like QEMU or virsh for management
        - Handles low-level virtualization directly on host hardware
    - *QEMU (Quick Emulator)*

- User-space application that emulates full hardware systems
- Can run virtual machines without KVM
- Offers better performance when paired with KVM
- Emulates components including
  - CPU
  - Hard drive
  - USB controller
  - Network card
- Responsible for building and managing virtual machines
- Uses KVM for hardware acceleration and performance
- Configures environment with options for
  - RAM allocation
  - Disk images
  - Network interfaces
  - Graphical output

- Key Differences
  - KVM provides core virtualization but no VM management
  - QEMU manages VM environment and can function independently
  - QEMU with KVM offers full virtualization with enhanced performance
- Summary
  - KVM enables Linux to act as a Type 1 hypervisor for CPU and memory virtualization
  - QEMU builds and runs virtual machines and emulates hardware
  - QEMU with KVM is a common setup for powerful and efficient Linux virtualization

- **VM Architecture and Performance**
    - Purpose of VM Architecture and Performance
        - Improve virtualization speed and scalability
        - Reduce emulation overhead
        - Support complex testing and development environments
    - *VirtIO*
        - Framework for optimized interaction between VM and host hardware
        - Replaces slow, full hardware emulation
        - Provides paravirtualized interfaces
        - Enables low-overhead communication
        - Supported device types
            - virtio-net for networking
            - virtio-blk for disk access
            - virtio-scsi for advanced storage
        - Works with KVM and QEMU environments
        - Reduces CPU load and increases I/O efficiency
        - Supported in modern distributions including RHEL Ubuntu and CentOS
        - Requires confirmation that VirtIO devices and drivers are active
    - *Paravirtualized Drivers*
        - Installed in the guest OS
        - Enable direct communication with VirtIO framework
        - Eliminate dependency on legacy emulated hardware
        - Common examples
            - virtio-net
            - virtio-blk
            - virtio-scsi

- virtio-rng
- Improve VM responsiveness scalability and efficiency
- Prevent fallback to emulated devices which degrade performance
- Administrators must ensure drivers are installed loaded and mapped correctly

- *Nested Virtualization*
  - Allows hypervisors to run inside a virtual machine
  - Enables VMs to host additional VMs
  - Useful in
    - Enterprise labs
    - Cloud simulation
    - Development and training environments
  - Introduces overhead and complexity
  - Requires monitoring of CPU and memory use
- Summary
  - VirtIO reduces emulation overhead by providing efficient virtual device interfaces
  - Paravirtualized drivers inside the guest OS are essential for leveraging VirtIO performance
  - Nested virtualization supports layered VM environments for advanced testing and infrastructure use
  - Combined these technologies provide scalable flexible high-performance virtualization in Linux

- **Operations**
    - Purpose of VM Operations
        - Understand virtual machine runtime states
        - Manage virtual disk image files
        - Ensure safe administration and efficient storage handling
    - *VM States*
        - Reflect the current activity of a virtual machine
        - Determine which administrative actions are allowed
        - Common states
            - running
                - VM is active and using system resources
            - paused
                - VM is temporarily halted without shutting down
            - shut off
                - VM is powered down but retains configuration and disk data
            - suspended
                - Memory contents saved to disk for later resumption
            - crashed
                - Indicates system failure due to error or misconfiguration
        - Tools like virsh used to monitor and manage these states
        - Administrative actions like disk resizing or hardware modification often require shut off or paused state
    - *Disk Image Operations*
        - Disk image is a file acting as the VM's virtual hard drive
        - Stores OS, applications, and user data

- Common operations
  - format conversion
    - Example
      - convert QCOW2 to RAW for compatibility or performance
  - resizing
    - Increases storage capacity for the VM
    - VM must be shut off before resizing
    - Guest OS may require partition and filesystem adjustment
  - inspecting properties
    - Displays format type, virtual size, actual usage, and backing files
- Disk image management supports
  - VM migration
  - environment consolidation
  - storage planning
  - disaster recovery
- Helps optimize provisioning automation and storage efficiency

- Summary
  - VM states define behavior and allowed administrative tasks
  - Disk image operations maintain virtual storage integrity and adaptability
  - Together these skills support scalable and efficient Linux virtualization environments

- **VM Resources**
  - Purpose of VM Resources

- Core components that affect VM performance scalability and connectivity
- Resources include CPU RAM storage and network interfaces
- Allocation determines how efficiently the host system supports each VM

- *CPU*
  - Virtual machines use vCPUs allocated from the host's physical CPUs
  - vCPUs are scheduled and managed by the hypervisor
  - Multiple VMs may share the same physical core
  - More vCPUs improve performance for CPU-intensive tasks
  - Overallocating vCPUs may harm overall host performance
  - Administrators must balance CPU usage across VMs
- *RAM*
  - RAM is short-term memory allocated to each VM from the host’s physical memory
  - Impacts how many applications can run and how the VM performs under load
  - Overcommitment may assign more virtual memory than physically available
  - Overcommitment is risky if multiple VMs use full memory simultaneously
  - Requires careful monitoring and adjustment based on workload
- *Storage*
  - Storage is provided through virtual disk image files on the host system
  - Disk image formats include
    - .qcow2
    - .raw
    - .vmdk
  - Disk image appears as a normal hard drive to the guest OS

- Benefits include resizing cloning snapshotting and portability
- Disk interface types affect performance
  - Examples
    - IDE
    - SATA
    - VirtIO
- Host storage performance impacts VM read/write speed
- Proper disk allocation prevents slowdowns under heavy load

- *Network*
  - VMs use vNICs to connect to networks
  - vNICs are software-defined and appear as physical NICs in the VM
  - vNICs connect to virtual switches or bridges on the host
  - Common network configurations
    - *NAT*
      - Uses host IP for external access
    - *Bridged*
      - Appears as a separate device on LAN
    - *Isolated/Internal*
      - Restricts traffic to VMs and host
  - Network mode affects performance routing firewall and access
  - Proper configuration ensures connectivity and security
- Summary
  - CPU resources are managed as vCPUs and must be balanced to avoid contention
  - RAM must be carefully allocated to avoid overcommitment issues

    - Storage is provided via virtual disk images which are flexible but performance-sensitive
    - Network interfaces are configured using vNICs and various network modes based on requirements

- **VM Management**
    - Purpose of VM Management
        - Supports efficient deployment and maintenance of virtual machines
        - Key techniques include image templates cloning snapshots and migrations
        - Ensures consistency scalability high availability and operational flexibility
    - *Baseline Image Templates*
        - Pre-configured VM images with OS common software and settings
        - Used as a master copy for new VM deployments
        - Saves time and ensures consistency in large-scale environments
        - Formats include QCOW2 VMDK VDI
        - Includes metadata and system configurations
        - Supports enterprise compliance and rapid provisioning
    - *Cloning*
        - Duplicates an existing VM including disk image and settings
        - Creates identical VMs quickly for web servers or test systems
        - Used for disaster recovery training and production replication
        - Often done from a baseline template
        - Allows post-cloning customization for hostname or role
    - *Snapshots*

- Captures the full VM state including disk memory and configuration
- Used before major changes or testing
- Enables rollback if updates or configs fail
- Includes CPU RAM and network settings for full restoration
- Short-term recovery method not a replacement for backups
- Should be cleaned up regularly to reduce disk usage and prevent slowdowns

- *Migrations*
  - Moves VMs between physical hosts
  - Two types of migration
    - *Cold migration*
      - Performed while VM is powered off
      - Safe method for moving inactive systems
    - *Live migration*
      - Moves running VMs with minimal downtime
      - Transfers memory CPU and I/O in real time
      - Requires shared storage and compatible host systems
      - Used for load balancing maintenance and hardware failure resolution
- Summary
  - Baseline templates provide consistent starting points for VM creation
  - Cloning enables fast replication of known-good environments
  - Snapshots support quick recovery during updates and tests
  - Migrations ensure uptime and flexibility in host maintenance and scaling

- **Network Types**
  - Overview of Virtual Network Types
    - Virtual machine network types control VM connectivity with internet, host, and other VMs
    - Common types include
      - NAT
      - Bridged
      - Host-only
      - Routed
      - Open
    - Multiple network adapters can be used per VM for combined networking modes
  - *NAT (Network Address Translation)*
    - VM borrows host's internet connection
    - External network sees only the host, not the VM
    - Simple and secure for testing environments
    - Useful in enterprise labs or when external access to VM isn't needed
    - Requires no additional configuration in hypervisor
    - Port forwarding is needed for inbound access
    - Not suitable for hosting externally accessible services
  - *Bridged Networking*
    - VM is placed directly on the physical network
    - Gets its own IP address
    - Can communicate with all other physical and virtual devices on the same LAN
    - Suitable for simulating production environments

        - Useful for testing production-level services or workstations
        - Security risk if misconfigured or exposed
        - Should be secured like a physical machine
    - *Host-only Networking*
        - VM can only communicate with host and other VMs in Host-only mode
        - No internet or LAN access
        - Best for isolated software testing or internal service simulation
        - Ideal for training and internal lab environments
        - No exposure to external threats
        - Cannot connect to update servers or external services
    - *Routed Mode*
        - VM accesses external networks through a virtual router
        - Allows custom routing rules
        - Useful for testing segmented networks or DMZ simulations
        - Requires careful configuration to prevent routing errors
        - Risk of misconfiguration leading to unintended traffic exposure
        - Best suited for experienced network administrators
    - *Open Networking (Promiscuous Mode)*
        - VM can view and interact with all network traffic
        - Used in advanced labs like penetration testing or network sniffing
        - Provides full network visibility
        - High risk if compromised or misused
        - Can snoop on traffic or affect network security
        - Should be tightly controlled or restricted in enterprise environments
    - Summary
        - NAT mode enables internet access via host but limits external reach

        - Bridged mode connects VM directly to the LAN with full network presence
        - Host-only creates an isolated internal network among VMs and host
        - Routed allows network simulation with virtual routing paths
        - Open grants full access and visibility, intended for specialized testing
        - VMs can use multiple adapters to mix connectivity modes for flexible lab setups

- **VM Tools**
    - Overview of VM Tools
        - Administrators need to manage many hypervisors with varying interfaces
        - libvirt provides a consistent management interface for hypervisors
        - virsh is a command-line tool that uses libvirt's API
        - virt-manager is a graphical interface that also uses libvirt
    - *libvirt*
        - Open-source library, daemon, and API for hypervisor management
        - Supports hypervisors such as
            - KVM/QEMU
            - Xen
            - LXC
            - OpenVZ
            - VirtualBox
            - VMware ESXi
        - Allows consistent management without learning new tools per platform

- Handles starting and stopping guests, snapshotting, migrating, resource tracking
- Operates through API requests which are processed by the libvirtd daemon
- Example of defining and enabling a directory-based storage pool
    - Command
        - sudo virsh pool-define-as local-dir --type dir --target /var/lib/libvirt/images
            - Defines a pool called local-dir in the given directory
        - sudo virsh pool-build local-dir
            - Ensures the directory exists on disk
        - sudo virsh pool-start local-dir
            - Activates the pool for immediate use
        - sudo virsh pool-autostart local-dir
            - Ensures the pool starts at boot

- *virsh*
    - Command-line interface to interact with libvirt
    - Script-friendly tool for VM management across platforms
    - Subcommands directly map to libvirt API functions
        - Command
            - virsh help
                - Lists all available subcommands
            - virsh list --all
                - Displays all defined VMs
            - virsh shutdown demo-vm
                - Shuts down a specific VM

        - virsh start demo-vm
            - Starts a specific VM
        - virsh reboot demo-vm
            - Reboots a VM
        - virsh save demo-vm /tmp/demo.state
            - Saves VM's state to disk for later resumption
        - virsh create /tmp/demo.xml
            - Launches a VM from XML configuration
    - Compatible with different hypervisors like KVM and Xen
    - Allows universal command sets across platforms
- *virt-manager*
    - Desktop graphical interface for libvirt
    - Simplifies VM management with point-and-click options
    - Mirrors functionality of virsh using the same backend
    - Tasks performed in GUI reflect in command-line and vice versa
    - Example usage
        - Click File → New Virtual Machine
        - Choose installer ISO
        - Assign 2 vCPUs and 4 GB RAM
        - virt-manager creates equivalent XML and issues virsh define and virsh start
    - Performance tab shows live statistics (CPU, disk, network) from libvirt
    - Ideal for administrators preferring GUI-based management
- Summary
    - libvirt standardizes VM management across multiple hypervisors
    - virsh enables command-line control through libvirt's API

- virt-manager offers a GUI with equivalent functionality to virsh
- All tools work together to provide flexible, efficient virtual machine administration

Containers

Objective 2.6: *Given a scenario, manage applications in a container on a Linux server*

- **Runtimes**
  - *Container*
    - Lightweight portable environment
    - Packages application and dependencies
    - Behaves the same across systems
  - *Container Runtimes*
    - Create and manage containers
    - Handle tasks like startup, isolation, resource control
    - Provide different levels of control and user interaction
  - *runC*
    - Low-level command-line tool
    - Follows Open Container Initiative (OCI) standard
    - Requires config.json and root filesystem
    - Used with runc run command
    - Not common for production workloads
    - Useful for troubleshooting and custom builds
  - *containerd*
    - Full container lifecycle management daemon
    - Uses runC for container execution
    - Handles image pulls, storage, networking
    - Runs in background as service

- Used by orchestration tools like Kubernetes
- Not commonly used directly via CLI

- *Docker*
  - All-in-one container platform
  - Includes CLI and REST API
  - Built on top of containerd and runC
  - docker pull and docker push
    - Manage image registries
  - docker run
    - Starts containers
  - docker ps
    - Lists running containers
  - docker rm
    - Removes containers
  - docker rmi
    - Deletes images
  - Used for CI/CD, testing, and service management
- *Podman*
  - Docker-compatible container engine
  - No central daemon
  - Supports rootless containers
  - podman run and podman ps mirror Docker CLI
  - Integrates with systemd for service management
  - Preferred in environments emphasizing security and compliance
- Summary
  - runC provides raw container execution based on OCI

        - containerd offers full container lifecycle as a daemon
        - Docker adds usability via CLI and API
        - Podman delivers Docker compatibility without daemon or root access

- **Building an Image**
    - Overview
        - Building a container image involves defining a base, user, entrypoint, and optional default command
        - Dockerfile instructions include FROM, USER, ENTRYPOINT, and CMD
    - *FROM*
        - FROM sets the base image for the container
        - Required as the first instruction in a Dockerfile
        - Defines the operating system and environment foundation
        - Example
            - FROM python:3.11-slim
                - Uses a lightweight Debian-based image with Python 3.11
    - *USER*
        - USER sets which user runs the container processes
        - Docker defaults to root, which can be a security risk
        - Non-root users can be created and used with RUN and USER
        - Example
            - RUN useradd -m appuser
            - USER appuser
                - Sets the user to appuser for all following commands
    - *ENTRYPOINT*

- ENTRYPOINT defines the primary command the container always runs
- Used to enforce consistent container behavior
- Example
    - ENTRYPOINT ["python", "app.py"]
        - Always runs python app.py when container starts

- *CMD*
    - CMD supplies default arguments to ENTRYPOINT or acts as a fallback command
    - Can be overridden at runtime
    - Example
        - CMD ["--debug"]
            - Appends --debug to ENTRYPOINT unless overridden
- Summary
    - FROM defines the base image and is the first required line
    - USER switches execution from root to a non-root user for security
    - ENTRYPOINT defines the container's main action
    - CMD provides default or overrideable arguments
    - Together, these Dockerfile instructions build secure and predictable images

- **Image Retrieval and Maintenance**
    - Overview
        - Container images include application code and dependencies
        - Images are pulled, tagged, layered, and cleaned up through pruning
        - Proper management ensures system efficiency and saves disk space

- *Pulling Images*
  - Downloads images from remote registries (e.g., Docker Hub)
  - Syntax
    - docker pull <image-name>[:tag]
  - If no tag is provided, defaults to latest
  - Example
    - docker pull ubuntu:20.04
  - Downloads the ubuntu image version 20.04 to the local system
- *Tags*
  - Identify versions or variants of container images
  - Syntax
    - <image-name>:<tag>
  - Example
    - docker pull nginx:latest
  - Download the latest published version of the nginx image
  - Example
    - docker pull nginx:1.21
  - Downloads a specific version of nginx
- *Layers*
  - Images are built from multiple layers
  - Each Dockerfile instruction creates a new layer
  - Layers are reused across images to save space and speed up builds
  - Example command
    - docker pull python:3.11-slim
  - Sample layer output
    - f7ec5a41d630: Pull complete

        - 3b2b1a174845: Pull complete
        - 5b3f5f3d0eb0: Pull complete
    - Already-present layers are skipped during pull
- *Pruning*
    - Cleans up unused containers, images, networks, and build cache
    - Syntax
        - docker system prune
    - Use -f to skip confirmation
    - Example
        - docker system prune -f
    - Sample output
        - Deleted containers: abc123456789, def987654321
        - Deleted images: untagged ubuntu:old, sha256:1b2...a4f
        - Total reclaimed space: 345MB
- Summary
    - docker pull fetches image layers from remote registries
    - Tags identify specific image versions for consistency
    - Images consist of reusable layers that optimize space and build time
    - docker system prune reclaims space by removing unused resources

- **Container Lifecyle Management**
    - Overview
        - Containers go through phases: create, run, stop, delete, and cleanup
        - Managing lifecycle ensures efficient use of resources and system cleanliness

- *Run*
  - Used to create and start a new container from an image
  - Syntax
    - docker run [options] <image-name>
  - Example
    - docker run -it ubuntu:20.04 bash
  - -it provides an interactive terminal
  - ubuntu:20.04 is the image being used
  - bash is the command run inside the container
  - Results in an interactive prompt such as root@container-id:/#
- *Start and Stop*
  - Controls the state of existing containers
  - Syntax
    - docker start <container-name>
    - docker stop <container-name>
  - Example
    - docker stop web-test
  - Gracefully halts the container's main process
  - Example
    - docker start web-test
  - Resumes the previously stopped container
- *Delete*
  - Removes a container permanently
  - Container must be stopped before deletion
  - Syntax
    - docker rm <container-name or ID>

        - Example
            - docker rm web-test
        - Frees disk space and reduces clutter
        - Fails if attempted on a running container
    - *Prune*
        - Cleans up all unused containers, images, and other resources
        - Syntax
            - docker system prune
        - Add -f to skip confirmation
        - Example
            - docker system prune -f
        - Deletes stopped containers, dangling images, unused networks, and build cache
        - Helps reclaim disk space and maintain order
    - Summary
        - docker run creates and starts a container
        - docker start and docker stop control container runtime state
        - docker rm deletes containers that are no longer needed
        - docker system prune removes all unused container resources to save space

- **Container Inspection and Interaction**
    - *Environmental Variables*
        - Used to pass configuration settings into a container at the time it starts

- These variables can control application behavior, set authentication credentials, or define runtime values without changing the container image itself
- The general syntax is
  - docker run -e <KEY>=<VALUE> <image-name>
- Example
  - docker run -e NODE_ENV=production node:18
- docker is the tool
- run is the action
- -e introduces the environment variable
- NODE_ENV is the key
- production is the value
- node:18 is the image being used
- Inside the container, the application can access NODE_ENV as a standard environment variable

- *Read Container Logs*
  - Reading container logs allows you to see the output of the container's main process
  - The logs show standard output (stdout) and standard error (stderr)
  - The general syntax is
    - docker logs <container-name or ID>
  - Example
    - docker logs web-server
  - docker is the tool
  - logs is the action
  - web-server is the name of the container

- This command displays the log output
- Example output
  - Server started on port 8080
  - Connected to database
  - Listening for requests

- *Inspect Containers*
  - docker inspect displays detailed configuration of a container in JSON format
  - The output includes data such as
    - Environment variables
    - Network settings
    - Mounts
    - Runtime state
  - The general syntax is
    - docker inspect <container-name or ID>
  - Example
    - docker inspect api-backend
  - docker is the tool
  - inspect is the action
  - api-backend is the container being inspected
  - Example output may include
    - "Env": [

      "NODE_ENV=production",

      "PORT=3000"

      ]

      "IPAddress": "172.17.0.3"

"Mounts": []
"State": {
"Status": "running"
}

- *Exec*
  - docker exec runs a command directly inside a running container
  - This allows you to troubleshoot or explore the container's filesystem
  - The general syntax is
    - docker exec -it <container-name or ID> <command>
  - Example
    - docker exec -it test-env bash
  - docker is the tool
  - exec is the action to run a command inside a container
  - -it allows interactive mode
  - test-env is the name of the container
  - bash is the command being run inside
  - After running, you are dropped into the container's shell prompt
- Summary
  - Environmental variables are used to pass configuration into containers at startup
  - Reading container logs helps monitor the container's activity
  - Inspecting containers provides detailed runtime and configuration data in JSON format
  - Exec allows direct interaction with running containers for debugging or administrative tasks

- **Container Storage and Tagging**
    - Overview
        - Managing image versions and persistent data is essential for container reliability
        - Tags identify image versions
        - Volume mapping ensures data persistence across container sessions
    - *Tags*
        - Tags label specific versions of a container image
        - Syntax
            - <image-name>:<tag>
        - Default tag is latest if not specified
        - Example
            - docker pull nginx:1.21
        - docker is the tool
        - pull is the action
        - nginx is the image
        - 1.21 is the version tag
        - Tags ensure the correct application version is pulled or run
    - *Map Container Volumes*
        - Links a host folder to a container folder
        - Ensures data persists even if the container is removed
        - Syntax
            - docker run -v <host-path>:<container-path> <image-name>
        - Example
            - docker run -v /home/user/data:/app/data ubuntu
        - docker is the tool

        - run is the action
        - -v is the volume flag
        - /home/user/data is the host directory
        - /app/data is the container directory
        - ubuntu is the image used
        - Files written to /app/data appear in /home/user/data on the host
        - Mapping is essential for persisting logs, uploads, or database files
    - Summary
        - Tags allow administrators to pull and run specific versions of container images
        - Volume mapping links host directories to container paths for persistent storage
        - Using these tools ensures consistent versioning and durable data handling

- **Volume Management Operations**
    - Overview
        - Volumes provide persistent storage for containers
        - Managed independently of containers
        - Data stored in volumes remains after container deletion
    - *Create Volume*
        - Command syntax
            - docker volume create <volume-name>
        - Example
            - docker volume create app-data
        - docker is the tool

- volume create is the action
- app-data is the name of the volume
- Creates a standalone, reusable data volume

- *Mapping Volume*
  - Connects volume to container path at runtime
  - Command syntax
    - docker run -v <volume-name>:<container-path> <image-name>
  - Example
    - docker run -v app-data:/data ubuntu
  - -v maps app-data to /data inside the container
  - ubuntu is the base image used to launch the container
  - Files written to /data persist in app-data volume
  - Volume data survives container removal
- *Pruning*
  - Removes unused (dangling) volumes
  - Frees up disk space
  - Command syntax
    - docker volume prune
  - -f flag skips confirmation prompt
  - Helps maintain system cleanliness and efficiency
- Summary
  - docker volume create sets up a persistent data volume
  - docker run -v maps the volume to a container directory
  - docker volume prune deletes unconnected volumes to recover space
  - These steps ensure reliable, persistent, and maintainable data storage in containerized environments

- **Volume Security and Filesystem Operations**
    - Overview of Volume Security and Filesystem Operations
        - Containers use a layered filesystem for storage
        - Overlay filesystem enables writable appearance while preserving base layers
        - SELinux context controls access based on labeled attributes
        - Proper configuration ensures container functionality and security
    - *Overlay Filesystem*
        - Overlay filesystems give containers the illusion of a full writable filesystem
        - Base files reside in lowerdir (read-only)
        - Changes are written to upperdir (writable)
        - Kernel performs “copy-up” operation from lowerdir to upperdir when changes occur
        - workdir is used to track copy-up operations
            - Command
                - mount | grep overlay
                    - Confirms how container layers are mounted
            - Example output
                - overlay on /var/lib/docker/overlay2/abc123/merged type overlay (rw,relatime,lowerdir=...,upperdir=...,workdir=...)
                    - rw
                        - Writable merged view
                    - relatime

- Updates access time only on file modification
- lowerdir
  - Base image (read-only)
- upperdir
  - Writable layer for changes
- workdir
  - Scratch space for copy operations
- Ensures container performance, integrity, and storage isolation

- *SELinux Context*
  - SELinux context uses labeled attributes to manage file access
  - Context includes four fields
    - User
      - Role
      - Type
      - Level
  - type field is most critical for containers
  - Command to view SELinux context
    - Command
      - ls -Z
    - Example output
      - -rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 index.html
        - unconfined_u
          - User

        - object_r
            - Role
        - httpd_sys_content_t
            - Type
        - s0
            - Level
    - Type controls process access to files
    - httpd_sys_content_t allows web servers like Apache or Nginx to access files
    - Incorrect types like default_t or container_file_t may restrict access unless explicitly allowed
- *Temporary SELinux Context Change*
    - Use chcon to change file or directory context temporarily
        - Command
            - chcon -t <type> <file-or-directory>
        - Example
            - chcon -t container_file_t /data
                - Assigns container-appropriate type to /data
                - Grants read/write access per SELinux policy
- *Permanent SELinux Context Change*
    - Use semanage followed by restorecon for permanent changes
        - Command
            - semanage fcontext -a -t <type> <path>
        - Example
            - semanage fcontext -a -t container_file_t '/data(/.*)?'
                - Sets type for /data and all contents

- Command
  - restorecon -Rv /data
    - Applies the correct context throughout the directory
- Ensures context persists across reboots and system relabels

- Summary
  - Overlay filesystem combines read-only and writable layers to manage container storage
  - Base layers remain unchanged while changes are saved in the top writable layer
  - SELinux context enforces file access rules through labeled attributes
  - Administrators can use chcon for temporary changes and semanage + restorecon for permanent configuration
  - Proper use of overlay and SELinux ensures secure and efficient container operations

- **Network Management Operations**
  - Overview
    - Containers need shared network environments to communicate
    - External access is controlled using port mapping
    - Docker provides built-in commands to manage networks and ports
  - *Create Network*
    - Virtual network connects containers securely and efficiently
    - Command syntax
      - docker network create [OPTIONS] NETWORK_NAME

- Example
  - docker network create --driver bridge webapp-net
- bridge is the default driver for single-host container networks
- Name (e.g., webapp-net) helps identify network
- Successful creation returns a unique network ID
- Internally Docker configures routing and NAT
- Containers can reach each other by name on the same network

- *Port Mapping*
  - Publishes internal container ports to host system
  - Command syntax
    - docker run -p HOST_PORT:CONTAINER_PORT IMAGE_NAME
  - Example
    - docker run -d -p 8080:80 --name webserver nginx
  - -d runs the container in detached mode
  - -p maps host port 8080 to container port 80
  - --name assigns container a recognizable name (webserver)
  - nginx is the base image for web server
  - Visiting http://localhost:8080 connects to containerized nginx server
  - Docker returns a container ID upon successful start
- Summary
  - docker network create builds private container networks
  - Enables internal name-based container communication
  - docker run -p maps host ports to container ports for external access
  - Proper network and port setup ensures secure, connected, and accessible container environments

- **Local Networks**
  - *bridge*
    - Default local network mode for Docker containers on a single host
    - Creates an isolated virtual network for internal container communication
    - Uses NAT (Network Address Translation) to allow external access via mapped ports
    - Command Syntax
      - docker run -d --network bridge -p HOST_PORT:CONTAINER_PORT IMAGE_NAME
    - Example
      - docker run -d --network bridge -p 8080:80 --name my-web nginx
    - Details
      - -d runs container in detached mode
      - --network bridge attaches the container to bridge network
      - -p 8080:80 maps host port 8080 to container port 80
      - NAT enables external users to access the service via http://localhost:8080
  - *host*
    - Shares the host system's network stack directly with the container
    - No NAT or port mapping required
    - Container uses host's IP and open ports
    - Offers better performance but removes network isolation
    - Command Syntax
      - docker run -d --network host IMAGE_NAME
    - Example
      - docker run -d --network host --name web-host nginx

        - Details
            - Container serves content directly on host's port 80
            - Useful for performance-sensitive applications needing full host network access
    - *none*
        - Disables all networking for the container
        - No access to host, other containers, or external systems
        - Ideal for security-focused or offline workloads
        - Command Syntax
            - docker run -d --network none IMAGE_NAME
        - Example
            - docker run -d --network none --name isolated busybox sleep 3600
        - Details
            - Container runs without any external or internal network connectivity
            - Only loopback interface is available
            - Suited for tasks like file processing or sandboxed operations
    - Summary
        - bridge: Enables inter-container communication and external access via NAT and port mapping
        - host: Shares host network stack; no isolation but improved performance
        - none: Disables networking entirely for full isolation

- **Advanced and Overlay Networks**
    - *IPvlan*

- IPvlan allows containers to use IP addresses from the same subnet as the host
- Containers share the host's physical network interface but use logical isolation
- Only one MAC address is used (the host's MAC), ensuring compatibility with most environments
- Useful for integrating containers into the physical network without NAT
- Administrator configures a subnet, gateway, and links the network to the host's interface
- Containers launched on the IPvlan network receive static IPs from the assigned subnet
- Containers communicate independently from the host while sharing the same interface

- *Macvlan*
  - Macvlan provides each container with its own MAC address and IP address
  - Containers appear as individual network nodes on the physical network
  - Offers compatibility with systems requiring unique MACs or legacy applications
  - No NAT is used, enabling direct external communication
  - Administrator sets a subnet, gateway, and specifies the physical network interface
  - Containers on the Macvlan network are externally reachable and fully visible on the network
- *Overlay*

- Overlay networks enable communication between containers on different Docker hosts
- Commonly used in clustered or orchestrated environments such as Docker Swarm
- Creates a virtual encrypted network that overlays the existing host infrastructure
- Does not rely on a single physical interface
- Facilitates secure and seamless container-to-container communication across nodes
- Overlay networks span the entire cluster and simplify distributed application deployment

- Summary
  - IPvlan allows containers to share the host's interface and subnet using the host's MAC address, while remaining logically isolated
  - Macvlan assigns containers their own MAC and IP addresses, making them behave like separate physical machines on the network
  - Overlay creates an abstract, encrypted network across multiple hosts, ideal for distributed or orchestrated container environments

- **Privileged vs Unprivileged**
  - Overview of Privileged vs Unprivileged Network Operations
    - Managing access to system resources affects both functionality and security
    - Focus is on how privileges impact networking tasks and kernel module management

- Covers security concepts, technical differences, and practical implementation

- Security Concepts
  - *Privileged operations*
    - Require elevated permissions such as root access
    - Examples of privileged tasks include
      - Configuring network interfaces
      - Changing routing tables
      - Loading kernel modules
  - *Unprivileged operations*
    - Are limited to safe user-space tasks
    - Cannot modify core system or network settings
    - Reduces chance of accidental misconfiguration or security abuse
    - Based on principle of least privilege
      - Users and processes get only minimum access necessary
      - Reduces attack surface and improves system security
- Technical Differences
  - Privileged users or containers can perform low-level network functions
    - Create raw sockets
    - Modify iptables rules
    - Access kernel interfaces like /sys and /proc/net
  - Unprivileged users or containers lack these permissions
  - Access limited to pre-defined virtual interfaces or user-space tools
  - Isolation is enforced through
    - Linux capabilities
    - Namespace configuration (e.g. user namespaces)

- Practical Implementation
  - Privileged network access is managed through
    - User roles
    - System configuration
    - Container runtime settings
  - Examples
    - Launching container with full network access may use elevated flags or host networking
    - Unprivileged containers use virtual bridges or user-space networking
  - Tools and permissions
    - ip, iptables, modprobe require root access
    - Commands like ping, ip a may be used by unprivileged users depending on system configuration
  - Administrators must
    - Understand which tools require elevated privileges
    - Assign or restrict access accordingly
    - Balance security and functionality when managing networking
- Summary
  - Privileged operations involve tasks that directly impact system or kernel behavior
  - Unprivileged operations are designed to limit risk and prevent unauthorized changes
  - Enforced through Linux capabilities and namespace isolation
  - Proper configuration helps administrators maintain both security and stability

- Kernel module management is a key area where privilege separation is critical

Network Services and Configurations

Objective 1.4: *Given a scenario, manage network services and configurations on a Linux server*

- **Network Configuration**
  - Overview
    - Linux resolves hostnames using a layered approach
    - Name resolution depends on /etc/hosts, /etc/resolv.conf, and /etc/nsswitch.conf
  - */etc/hosts*
    - /etc/hosts provides manual hostname-to-IP mappings
    - Used for local resolution without relying on DNS
    - Helpful in isolated environments or when DNS is unavailable
    - Used for troubleshooting DNS issues by overriding with manual entries
    - Example entry
      - 192.168.1.50 test.local
        - Associates test.local with 192.168.1.50
          - System checks /etc/hosts before DNS if ordered accordingly in nsswitch.conf
  - */etc/resolv.conf*
    - /etc/resolv.conf specifies which DNS servers to use
    - Contains lines beginning with nameserver followed by an IP address
    - Typically points to local router or public DNS like 8.8.8.8
    - Crucial for troubleshooting internet name resolution
    - Example

        - nameserver 8.8.8.8
    - */etc/nsswitch.conf*
        - /etc/nsswitch.conf controls lookup order for resolving names and other data
        - Determines sequence of checking /etc/hosts and DNS
        - Common configuration
            - hosts: files dns
                - Checks /etc/hosts first, then DNS
        - If ordered as
            - hosts: dns files
                - System prioritizes DNS over local entries
        - Improper order may cause resolution failure even with correct entries in /etc/hosts
    - Summary
        - /etc/hosts provides manual hostname resolution
        - /etc/resolv.conf defines DNS servers for unresolved names
        - /etc/nsswitch.conf sets the order of name resolution sources
        - Correct configuration of all three is key to reliable network name resolution

- **NetworkManager**
    - Purpose of NetworkManager
        - Manages network connections across interfaces like Wi-Fi Ethernet and VPN
        - Keeps systems connected and handles switching between networks

- Provides command-line and graphical tools for network configuration

- *nmcli*
  - Primary command-line tool for interacting with NetworkManager
  - Used to monitor and manage connections
  - Common subcommands
    - general status
    - connection show
    - connection up <profile-name>
    - connection down <profile-name>
    - connection edit
    - device status
  - Example
    - nmcli connection up Wired-Profile
  - Offers real-time control for activation deactivation and modification of connections
  - Useful for administrators troubleshooting or configuring networks without a GUI
- *nmconnect (nm-connection-editor)*
  - Graphical interface for managing NetworkManager connection profiles
  - Run with command
    - nm-connection-editor
  - Allows configuration of
    - Static IP addresses
    - DHCP settings
    - DNS servers
    - IPv6 options

        - Wireless security
        - Proxy settings
    - Changes saved to .nmconnection files
    - Files located at
        - /etc/NetworkManager/system-connections/
    - Automatically updated with any changes made via nmcli or nm-connection-editor
- *.nmconnection Files*
    - Contain persistent network profile configurations
    - Include details such as
        - Interface name
        - IP addressing method
        - DNS settings
        - Security configurations
- Summary
    - NetworkManager controls all network interfaces and ensures stable connectivity
    - nmcli provides full command-line management of network settings and profiles
    - nm-connection-editor provides a graphical alternative for configuring complex settings
    - All changes are stored in .nmconnection files under /etc/NetworkManager/system-connections/

- **Netplan**
  - Overview of Netplan
    - *Netplan*
      - Is the default tool used to configure and manage network settings in Linux
      - Replaces traditional tools like ifconfig or manual file editing
      - Uses simple YAML configuration files stored in /etc/netplan/
      - Applies changes through backends like NetworkManager or systemd-networkd
      - Makes configurations more consistent, structured, and automation-friendly
  - Configuration Files
    - Netplan configurations are written in YAML format
    - Stored in /etc/netplan/ directory
    - Define interface behavior
      - Static IP assignments
      - DHCP enablement
      - Gateways and DNS servers
    - Changes are not auto-applied
    - Must be activated using Netplan commands
    - Example YAML content for static IP on eth0

```
network:
  version: 2
  ethernets:
    eth0:
      addresses: [192.168.1.100/24]
      gateway4: 192.168.1.1
      nameservers:
        addresses: [8.8.8.8, 1.1.1.1]
```

- *netplan try*
  - Used to test new configuration with a temporary safety window
  - Applies changes for 120 seconds
  - Requires confirmation with confirm command
  - If unconfirmed, settings automatically revert
  - Prevents lockout, especially when working remotely
  - Example usage
    - After editing YAML file
    - Run: sudo netplan try
    - Output

```
Configuration accepted. Please confirm within 120 seconds by typing 'confirm'
```

      - If confirmed: changes persist
      - If not confirmed: system reverts to previous settings
- *netplan apply*
  - Permanently applies network settings
  - Reads YAML files in /etc/netplan/
  - Activates interfaces using defined renderer
  - Used after confirming settings with netplan try or manual verification
  - Example usage
    - After editing and validating YAML file
    - Run: sudo netplan apply
- *netplan status*
  - Displays active configuration status
  - Shows renderer, addresses, routes, DNS for each interface

        - Used to confirm applied settings
        - Example output

```
enp0s3: configured
renderer: networkd
addresses: [192.168.1.100/24]
routes: [default via 192.168.1.1]
dns: [8.8.8.8, 1.1.1.1]
```

    - Summary
        - Netplan centralizes network configuration using YAML files
        - Configuration files in /etc/netplan/ must be activated with commands
        - netplan try safely tests changes with rollback on failure
        - netplan apply commits settings permanently
        - netplan status verifies configuration is active and applied

- **IP Network Manangement Tool**
    - Purpose of IP Network Management Tool
        - View and manage network settings
        - Configure interfaces IP addresses and routing
        - Use modern ip command suite instead of deprecated ifconfig
    - *ifconfig*
        - Legacy command for viewing and configuring network interfaces
        - Still available on some systems via net-tools package
        - Usage
            - ifconfig
        - Example output
            - Interface name

        - eth0
      - IP address
        - inet 192.168.1.100
      - Netmask
        - 255.255.255.0
      - Broadcast
        - 192.168.1.255
      - MAC address
        - 00:1a:2b:3c:4d:5e
  - *ip address (ip a)*
    - Displays IP address configuration for interfaces
    - Supports both IPv4 and IPv6
    - Usage
      - ip address show
      - ip a
    - Example output
      - Interface
        - enp0s3
      - State
        - UP
      - Address
        - 192.168.1.100/24
      - Broadcast
        - 192.168.1.255
    - Useful for checking address assignments and operational state
  - *ip link (ip l)*

- Displays and configures link-layer settings of interfaces
- Shows MAC address MTU and link state
- Usage
  - ip link show
  - ip l
- Example output
  - Interface
    - enp0s3
  - State
    - UP
  - MAC address
    - 08:00:27:12:34:56
- Used to enable or disable interfaces
  - ip link set enp0s3 up

- *ip route (ip r)*
  - Displays and modifies the routing table
  - Helps with gateway configuration and route troubleshooting
  - Usage
    - ip route show
    - ip r
  - Example output
    - Default route
      - default via 192.168.1.1 dev enp0s3
    - Subnet route
      - 192.168.1.0/24 dev enp0s3 src 192.168.1.100
  - Used for diagnosing unreachable networks or incorrect routing

- Summary
  - ifconfig is deprecated but still used in legacy systems
  - ip address shows interface IP details
  - ip link displays link-layer status and allows interface control
  - ip route reveals how packets are routed from the system
  - The ip suite provides comprehensive control for Linux network management

- **Network Configuration Tools**
  - Overview of Network Configuration Tools
    - Network tools in Linux are essential for system identification, local address resolution, and interface diagnostics
    - Three core tools include
      - hostname
      - arp
      - ethtool
    - Each tool serves a unique purpose in managing or investigating networking behavior
  - *hostname*
    - Used to view or set the system's network name
    - Helps identify the system in logs, DNS queries, and prompts
    - Affects how the system is recognized by others on the network
    - Common options
      - hostnamectl set-hostname <name>
        - Sets hostname persistently on systems using systemd

- hostname -f
  - Displays the fully qualified domain name (FQDN)

- Example command

```
sudo hostnamectl set-hostname webserver01
```

- Sets hostname to webserver01
- Takes effect immediately and persists after reboot

- *arp*
  - Displays and modifies the Address Resolution Protocol (ARP) table
  - Maps IP addresses to MAC addresses on the local network
  - Automatically populated during communication with other devices
  - Common options
    - arp -a
      - Displays all ARP entries
    - arp -d <IP>
      - Deletes a specific ARP entry
    - arp -s <IP> <MAC>
      - Sets a static ARP entry
  - Example output from arp -a

```
router.local (192.168.1.1) at 00:11:22:33:44:55 [ether] on enp0s3
```

- Indicates router.local is at 192.168.1.1 with MAC 00:11:22:33:44:55 on interface enp0s3

  - Useful for troubleshooting IP conflicts or unreachable hosts
- *ethtool*
  - Used to inspect and configure Ethernet network interfaces

- Displays statistics, driver info, link status, and allows setting parameters
- Common options
  - -i
    - Shows driver and firmware information
  - -S
    - Displays interface statistics
  - -t
    - Runs diagnostic self-tests
  - -s
    - Sets speed and duplex
  - -f
    - Firmware operations
- Example command

```
sudo ethtool –S enp0s3
```

  - Returns traffic stats such as
    - rx_packets: 105432
    - tx_packets: 98231
    - rx_errors: 0
    - tx_errors: 0
- Helps monitor packet flow, errors, and assess interface reliability
- Additional examples
  - ethtool -i enp0s3 shows driver info
  - ethtool -t enp0s3 performs self-test

- Summary
  - hostname identifies the system by name on a network

        - arp shows MAC-to-IP address mappings for local communication
        - ethtool provides in-depth data and control for Ethernet interfaces
        - Together, these tools enable precise control and visibility of Linux network behavior

- **Network Connectivity Tools**
    - Purpose of Network Connectivity Tools
        - Diagnose connectivity and performance issues
        - Test reachability, trace network paths, and measure bandwidth
        - Useful for administrators to identify latency, packet loss, or routing problems
    - *ping and ping6*
        - Check basic host reachability and round-trip time
        - ping is for IPv4
        - ping6 is for IPv6
        - Use ICMP echo requests and replies
        - Syntax
            - ping [destination]
            - ping6 [destination]
        - Example
            - ping google.com
        - Output includes response time per packet and summary statistics
        - Helps identify packet loss and latency
    - *traceroute*
        - Displays each hop in the path to a destination

- Uses incrementing TTL to reveal routers along the route
- Syntax
  - traceroute [destination]
- Example
  - traceroute example.com
- Identifies where delays or failures occur between source and target

- *tracepath*
  - Similar to traceroute but does not require root privileges
  - Uses UDP and checks for MTU issues
  - Syntax
    - tracepath [destination]
  - Example
    - tracepath example.com
  - Useful for quick diagnostics without elevated access
- *mtr*
  - Combines ping and traceroute into a live updating interface
  - Displays packet loss and latency in real time per hop
  - Syntax
    - mtr [destination]
  - Example
    - mtr example.com
  - Interactive tool exited with q
  - Effective for spotting intermittent or changing network behavior
- *iperf3*
  - Measures actual bandwidth between two systems
  - Requires iperf3 installed and server set up on remote host

        - Uses TCP or UDP
        - Syntax
            - iperf3 -c [server IP]
        - Example output
            - [ID] Interval Transfer Bandwidth
            - 0.00–10.00 sec 1.10 GBytes 942 Mbits/sec
        - Helps evaluate real-world throughput and network capacity
    - Summary
        - ping and ping6 test host reachability and response time
        - traceroute maps full path to destination and identifies delays
        - tracepath provides similar output without root and checks MTU
        - mtr offers continuous live view of latency and packet loss
        - iperf3 tests bandwidth between two endpoints over TCP or UDP

- **Scanning and Traffic Analysis Tools**
    - Overview
        - Network analysis tools provide insight into ports, sockets, traffic, and service availability
        - Tools covered include ss, nc, tcpdump, and nmap
    - *ss*
        - Displays active and listening sockets on a Linux system
        - Syntax
            - ss [options] [FILTER]
        - Example
            - ss -lnt src :22

        - -l shows listening sockets
        - -n prevents DNS resolution
        - -t filters TCP only
        - src :22 shows source port 22
    - Sample output
        - LISTEN 0 128 0.0.0.0:22 0.0.0.0:*
            - Confirms SSH is listening on port 22 on all interfaces
- *nc (netcat)*
    - General-purpose utility for network connectivity and file transfers
    - Syntax
        - nc [options] host port
    - Connection test example
        - nc -vz -w2 192.168.1.10 80
            - -v verbose
            - -z probe without I/O
            - -w2 timeout after 2 seconds
    - Can also listen on a port using -lp
    - Options like -p specify the source port, -n disables DNS
- *tcpdump*
    - Captures and analyzes raw packets on a specified interface
    - Syntax
        - tcpdump [options] [expression]
    - Packet capture example
        - sudo tcpdump -I -n -v -w web_traffic.pcap -i wlan0 port 80
            - -I enables monitor mode for wireless
            - -n disables DNS resolution

        - -v verbose headers
        - -w writes capture to file
        - -i specifies interface
        - port 80 filters HTTP traffic
- *nmap*
    - Network scanner used to detect open, closed, or filtered ports
    - Syntax
        - nmap [options] target
    - Subnet scan example
        - sudo nmap -sS -p 22,80,443 -T4 -Pn 192.168.1.0/24
            - -sS performs SYN scan
            - -p limits port scan
            - -T4 uses faster timing
            - -Pn skips ping checks
    - Output interpretation
        - open
            - Service is accepting connections
        - closed
            - No service listening
        - filtered
            - Firewall blocked probe
- Summary
    - ss lists socket usage and listening ports for real-time status
    - nc creates or tests connections and can transfer files
    - tcpdump captures and saves traffic for inspection
    - nmap scans networks and identifies open, closed, or filtered services

        - Together, these tools enable deep visibility into traffic and connectivity issues

- **DNS Tools**
    - Overview
        - Linux DNS tools include nslookup, dig, and resolvectl
        - Each tool provides different levels of detail and control for DNS queries
    - *nslookup*
        - Command used to perform basic DNS lookups
        - Syntax
            - nslookup <name> [server]
        - Example
            - nslookup www.example.com 8.8.8.8
                - www.example.com
                    - Domain to resolve
                - 8.8.8.8
                    - DNS server used for the query
        - Outputs include server info and resolved address
        - Displays “Non-authoritative answer” when response is from a DNS cache or referral
        - Known for minimal, directory-style display
    - *dig*
        - Provides detailed DNS response and query information
        - Syntax
            - dig [@server] <name> [type] [options]

- Example
  - dig @1.1.1.1 example.com MX +noall +answer +stats
    - @1.1.1.1: specific DNS server (Cloudflare)
    - example.com: domain to query
    - MX: mail exchange record type
    - +noall: suppresses all sections
    - +answer +stats: re-enables specific sections
- Outputs include record type, TTL, query time, and server
- Useful for debugging delegation, DNSSEC, and latency
- Script-friendly format

- *resolvectl*
  - Queries the local systemd-resolved DNS stub
  - Syntax
    - resolvectl <verb> [arguments]
  - Example
    - resolvectl query www.example.com
  - Output includes
    - Resolved address
    - Interface used (e.g., eth0)
    - Protocol used
    - Round-trip time
    - DNSSEC status
  - Displays how local system settings influence DNS behavior
  - Useful for inspecting per-interface DNS config and resolver cache
- Summary
  - nslookup

- Minimal and fast, best for quick human-readable queries

- dig
  - Detailed output for debugging and automation
- resolvectl
  - Integrated with systemd, reveals local DNS resolver behavior and configuration
- Combined use of these tools enables full insight into both external DNS resolution and internal name lookup mechanics

Backup and Restore Operations

Objectives:

- 1.5 - Given a scenario, manage a Linux system using common shell operations
- 1.6 - Given a scenario, perform backup and restore operations for a Linux server

- **Relative Paths**
  - Purpose of Relative Paths
    - Enables navigation based on current directory
    - Avoids full absolute paths
    - Supports quicker file referencing and command execution
  - *. (Current Directory)*
    - Represents the folder currently in use
    - Used to run files or scripts in the current location
    - Example
      - ./backup.sh
    - Prevents system from searching only in directories listed in $PATH
    - Ensures local script or file is targeted directly
  - *.. (Parent Directory)*
    - Refers to the directory one level above the current one
    - Used for accessing files or folders from the immediate upper directory
    - Example
      - cp ../index.html.
    - First path uses .. to reference index.html in parent directory
    - Second path uses . to place the file in the current directory

- *~ (Home Directory)*
  - Refers to the user's home directory
  - Works from anywhere in the file system
  - Used for returning to or referencing files in the home directory
  - Example
    - nano ~/notes.txt
  - Simplifies file access by avoiding long path typing such as /home/username/
- Summary
  - . represents the current directory
  - .. refers to the parent directory
  - ~ points to the user's home directory
  - Relative paths improve efficiency and simplify command execution

- **Archiving**
  - Archiving Overview
    - Archiving combines multiple files into a single package
    - Simplifies storage, transfer, and organization
    - Common tools
      - tar
      - cpio
  - *tar Command*
    - Packages multiple files or directories into a single archive
    - Common options
    - -c create archive

- -x extract files
- -t list archive contents
- -v verbose output
- -r append files to archive
- -f specify archive file
- -z compress with gzip
- -j compress with bzip2

- Options can be combined under a single dash
- General syntax
  - tar -[options] archive_name.tar [file(s) or directory]
- Example
  - tar -czvf backup.tar.gz configs/
    - -c create
    - -z gzip compression
    - -v verbose
    - -f archive file name
    - configs/ is the target folder
- *cpio Command*
  - Gets file list from another command like find
  - Common modes
    - -o copy-out (create archive)
    - -i copy-in (extract files)
    - -v verbose output
    - -u overwrite files during extraction
  - Options must be written separately
  - General syntax to create archive

- find [files] | cpio -ov > archive.cpio
- Example to archive .conf files in /etc
  - find /etc -name "*.conf" | cpio -ov > etc-configs.cpio
- Example to extract
  - cpio -iuv < etc-configs.cpio
    - -i extract
    - -u overwrite
    - -v show progress

- Summary
  - Archiving simplifies file management
  - tar is all-in-one with compression and combined options
  - cpio works with external file lists and gives fine control
  - Understanding both provides flexibility in Linux file management

- **Compression Tools**
  - *gzip*
    - Compresses files into .gz format
    - Fast and commonly used for everyday compression tasks
    - Supports switch combinations after a single dash
    - Common options
      - -d
        - Decompress
      - -f
        - Force overwrite
      - -n

- Do not save filename/timestamp
- -N
  - Save filename/timestamp
- -q
  - Quiet mode
- -r
  - Recursive directory compression
- -v
  - Verbose output
- -t
  - Test file integrity

- Syntax
  - gzip [options] filename
- Example
  - gzip -vr /var/log/
    - Compresses files recursively in /var/log/ with verbose output

- *bzip2*
  - Compresses files into .bz2 format
  - Slower but provides better compression than gzip
  - Companion tools
    - bunzip2
      - Decompress
    - bzcat
      - View contents
    - bzip2recover

- Attempt recovery from damaged archive
- bzless and bzmore
  - Scroll through compressed content

- Syntax
  - bzip2 [options] filename
- Example
  - bzip2 -v backup.sql && bzcat backup.sql.bz2
  - Compresses file with verbose output, then views contents

- *xz*
  - Compresses files into .xz format
  - Highest compression, slower processing
  - Suitable for static long-term archiving
  - Switches can be combined with a single dash
  - Common options
    - -d
      - Decompress
    - -f
      - Force overwrite
    - -q
      - Quiet
    - -v
      - Verbose
    - -t
      - Test archive
  - Syntax
    - xz [options] filename

- Example
  - xz -v backup.sql
    - Compresses backup.sql with verbose output

- *7-Zip (7z)*
  - Versatile tool supporting .7z, .zip, .tar formats
  - Uses p7zip package in Linux
  - No dashes used for options
  - Common commands
    - a
      - Add files to archive
    - x
      - Extract
    - l
      - List contents
    - t
      - Test archive
    - d
      - Delete from archive
  - Syntax
    - 7z <command> [archive_name] [file(s)]
  - Example
    - 7z a etc_backup.7z /etc && 7z l etc_backup.7z
      - Compresses /etc and then lists archive contents
- Summary
  - gzip: fast, simple, widely supported (.gz)
  - bzip2: better compression than gzip (.bz2), slower

- xz: highest compression for archival (.xz), slowest
- 7z: versatile cross-format utility for compression and extraction (.7z, .zip, .tar)
- Each tool serves different use cases based on speed, compression ratio, or format compatibility

- **Decompression Tools**
  - *gunzip*
    - Decompresses .gz files created by gzip
    - Restores original file from compressed version
    - Common options
      - -f
        - Force decompression even if file exists
      - -v
        - Verbose mode
      - -k
        - Keep the original .gz file
      - -r
        - Decompress directories recursively
    - Options can be combined after a single dash
    - Syntax
      - gunzip [options] filename.gz
    - Example
      - gunzip -vk /var/log/syslog.gz
        - Decompresses file with progress and keeps original

- *bunzip2*
  - Decompresses .bz2 files created by bzip2
  - Deletes compressed file after extraction by default
  - Common options
    - -v
      - Verbose output
    - -f
      - Force overwrite
  - Syntax
    - bunzip2 [options] filename.bz2
  - Example
    - bunzip2 -v backup.sql.bz2
      - Decompresses file and shows progress
- *unxz*
  - Decompresses .xz files
  - Retains original file with -k
  - Suitable for large file archives
  - Common options
    - -k
      - Keep the original .xz file
    - -f
      - Force overwrite
    - -v
      - Verbose output
    - -t
      - Test archive integrity

- Options can be bundled after one dash
- Syntax
  - unxz [options] filename.xz
- Example
  - unxz -vk backup.sql.xz
    - Decompresses file, shows output, and keeps original

- *7z*
  - Supports multiple archive formats like .7z, .zip, .tar
  - Uses p7zip package on Linux
  - No dashes used in commands
  - Common commands
    - x
      - Extract with full paths
    - e
      - Extract to current directory
    - l
      - List archive contents
    - t
      - Test archive integrity
  - Syntax
    - 7z <command> [archive_name] [options]
  - Example
    - 7z x project_backup.7z
      - Extracts contents with folder structure preserved
- Summary
  - gunzip: decompresses .gz files, supports verbose and keep options

- bunzip2: decompresses .bz2 files, deletes compressed file by default
- unxz: decompresses .xz files, supports keeping the original
- 7z: flexible decompressor for multiple formats, uses command-based syntax
- These tools allow restoration of compressed data across various formats in Linux

- **Data Copy and Recovery**
  - *dd*
    - dd is a low-level data copying tool for creating byte-for-byte disk or partition images
    - Operates at the block level regardless of filesystem type
    - Stops on read errors unless conv=noerror,sync is used to continue and pad with zeros
    - Common options include
      - if=
        - Input file or device
      - of=
        - Output file or device
      - bs=
        - Block size
      - count=
        - Number of blocks to copy
      - status=
        - Controls progress reporting

- Syntax
  - dd if=<input_file> of=<output_file> [options]
- Example
  - dd if=/dev/sda of=/mnt/backup/disk.img bs=4M status=progress
    - Creates a block-level image of /dev/sda
    - Writes image to /mnt/backup/disk.img
    - Uses 4MB blocks and displays progress

- *ddrescue*
  - ddrescue is designed for recovering data from damaged disks
  - Skips over bad blocks and retries them later
  - Supports log files for progress tracking and resumption
  - Syntax
    - ddrescue [options] <input_file> <output_file> <log_file>
  - Example
    - ddrescue /dev/sdb corrupted.img rescue.log
    - Reads from /dev/sdb
    - Writes recoverable data to corrupted.img
    - Logs progress to rescue.log
    - Can resume recovery if interrupted
- *rsync*
  - rsync synchronizes files and directories efficiently
  - Copies only new or changed data
  - Supports local and remote synchronization
  - Preserves metadata such as permissions, ownership, and timestamps
  - Requires -r for recursive copy or -a (archive mode) to preserve attributes
  - Syntax

- rsync [options] source destination
- Example
  - rsync -avh /home/user/ /mnt/backup/user/
  - Recursively copies /home/user/ to /mnt/backup/user/
  - Preserves attributes (-a)
  - Shows verbose output (-v)
  - Displays human-readable file sizes (-h)

- Summary
  - dd creates exact block-level copies, suitable for imaging and duplication
  - ddrescue handles damaged disks by skipping unreadable sectors and logging progress
  - rsync efficiently copies and syncs files or directories, preserving attributes and minimizing data transfer
  - Each tool provides unique benefits for backup, recovery, and system cloning tasks in Linux environments

- **Compressed File Operations**
  - *zcat*
    - zcat displays the contents of .gz compressed files directly in the terminal
    - Does not require decompression before viewing
    - Works like cat but for compressed files
    - Common options include
      - -f
        - Forces output even if the file is not compressed

- -v
    - Enables verbose output
- Redirection operators > and |
    - Can be used to send output to files or pipelines

- Syntax
    - zcat [options] filename.gz
- Example
    - zcat -f /var/log/syslog.1.gz > extracted_syslog.txt
    - Outputs the contents of syslog.1.gz to extracted_syslog.txt

- *zless*
    - zless provides scrollable and interactive viewing of .gz compressed files
    - Functions like less for compressed content
    - Does not require manual decompression
    - Supports navigation and searching using standard less commands
    - Syntax
        - zless filename.gz
    - Example
        - zless /var/log/apache2/access.log.1.gz
        - Opens access.log.1.gz for scrolling and searching
        - Example search /GET to locate HTTP GET requests
- *zgrep*
    - zgrep searches for patterns inside .gz compressed files without decompression
    - Behaves like grep with temporary decompression behind the scenes
    - Common options include
        - -i

        - For case-insensitive search
    - -n
        - To show line numbers
    - -v
        - To invert the match
- Syntax
    - zgrep [options] "search_pattern" filename.gz
- Example
    - zgrep -i "failed password" /var/log/auth.log.1.gz
    - Searches for case-insensitive pattern “failed password” in auth.log.1.gz
    - Outputs matching lines

- Summary
    - zcat is used to view full content of .gz files without unpacking
    - zless provides interactive navigation through compressed content
    - zgrep allows pattern searching inside compressed files
    - These tools save time and storage when analyzing logs or troubleshooting compressed data

Authorization, Authentication, and Accounting Methods

Objective 3.1: *Summarize authorization, authentication, and accounting methods*

- **Local Authentication**
  - Overview
    - Managed by PAM and Polkit
    - PAM handles identity verification and authentication policies
    - Polkit controls authorization for privileged actions after login
  - *PAM (Pluggable Authentication Modules)*
    - Framework for user authentication in Linux
    - Used by login, sudo, passwd, and other services
    - Configuration files stored in /etc/pam.d/
    - Modules operate under four interfaces
      - auth
        - Verifies user identity (e.g., password checking)
      - account
        - Enforces access policies (e.g., account expiration)
      - password
        - Handles password changes
      - session
        - Manages actions at login/logout (e.g., logging)
    - PAM Control Flags
      - required
        - Must pass, processing continues even if it fails

- requisite
  - Must pass, failure causes immediate termination
- sufficient
  - Success means authentication may succeed early if no required module failed
- optional
  - Only evaluated if it's the only module in the group

- Example PAM Usage
  - pam_pwquality.so
    - Enforces password complexity
  - Password change example
    - passwd username
  - Enforces rules from
    - /etc/pam.d/common-password
    - /etc/security/pwquality.conf
  - Example prompt

```
New password:
BAD PASSWORD: it is based on a dictionary word
Retype new password:
passwd: all authentication tokens updated successfully
```

- PAM Debugging
  - pam_tally2
    - View failed login attempts and unlock accounts

- *Polkit (PolicyKit)*
  - Manages fine-grained authorization for system actions
  - Files stored in
    - /etc/polkit-1/rules.d/

        - JavaScript-style rules
    - /etc/polkit-1/localauthority/
        - .pkla files
- Used for authorizing actions without sudo
- Polkit Commands
    - pkexec /usr/bin/apt update
        - Asks Polkit to allow running as root
    - pkaction --verbose
        - Lists available Polkit actions and policies
    - pkcheck --action-id [ID] --allow-user-interaction
        - Tests if current user is authorized for a specific action
    - pkttyagent --process $$
        - Registers authentication agent in terminal for GUI or SSH sessions

- Summary
    - PAM handles user authentication
        - Identity verification, login control, and password policies
    - Polkit handles authorization
        - Determines if users can perform privileged operations post-authentication
    - PAM
        - Front door security
    - Polkit
        - Room-by-room access control
    - Combined, they provide layered local security on Linux systems

- **Directory-based Identity Management**
  - *Kerberos*
    - Kerberos is a secure network authentication protocol
    - Users and services prove identity using encrypted tickets
    - Prevents password transmission over the network
    - Relies on a trusted Key Distribution Center (KDC)
    - Integrated with PAM for user logins in Linux
    - Common in environments like Active Directory or MIT Kerberos
    - Example
      - User logs in → PAM contacts KDC → Ticket-Granting Ticket (TGT) issued → User authenticated
  - *LDAP*
    - LDAP stands for Lightweight Directory Access Protocol
    - Standardized protocol for accessing directory data
    - Stores user accounts, group memberships, and organizational units
    - Often combined with Kerberos for authentication
    - Used as the backend identity store in enterprise systems
    - Integrated with PAM for centralized login capability
    - Simplifies user management across multiple systems
    - Example
      - LDAP client config added to system → Centralized user logs in → No local user account needed
  - *SSSD / Winbind*
    - SSSD stands for System Security Services Daemon
    - Winbind is part of the Samba suite
    - Both enable Linux to connect to centralized identity sources

- Act as local caches and connectors to LDAP, Kerberos, or Active Directory
- SSSD is modern and works with both LDAP and Kerberos
- Integrated with PAM for authentication
- Winbind commonly used for Active Directory integration
- Admin tasks include installing packages and configuring
  - /etc/sssd/sssd.conf for SSSD
  - /etc/samba/smb.conf for Winbind
- Once configured
  - Central directory users can log in
  - Home directories can be created automatically
  - Users treated like local accounts

- Summary
  - Kerberos provides secure, ticket-based authentication
  - LDAP stores directory data like users and groups for central access
  - SSSD and Winbind connect Linux to centralized identity systems
  - Together, they support unified login and access control
  - Enable consistent, secure identity management in enterprise Linux environments

- **Network / Domain Integration**
  - *realm*
    - Used to integrate Linux into enterprise domain environments
    - Focuses on authentication and domain membership
    - Simplifies configuration by automating SSSD, Kerberos, and LDAP setup
    - Allows Linux to join Windows Active Directory (AD) domains

- Enables domain users to authenticate on Linux systems with existing credentials
- Example command
  - sudo realm join --user=administrator corp.example.com
- Additional commands
  - realm list to verify domain membership
  - realm permit to allow domain users or groups to log in
- Does not handle file or printer sharing
- Used for identity and login integration with centralized domain systems

- *Samba*
  - Enables file and printer sharing between Linux and Windows systems
  - Allows Linux to access Windows shares or act as a file server for Windows clients
  - Can join Active Directory domains
  - Capable of functioning as a domain controller in some configurations
  - Provides Windows-compatible network services
  - Admin configuration includes editing smb.conf
  - Example configuration
    - [shared]

      path = /srv/shared

      read only = no

      browsable = yes
  - Service started using sudo systemctl start smbd
  - Shared folder can be accessed by Windows users via Linux server hostname or IP
  - Focuses on interoperability at the network services level

- Summary
  - realm joins Linux to AD domains for centralized identity and authentication
  - Samba enables file and printer interoperability with Windows systems
  - realm and Samba are complementary for enterprise Linux-Windows integration
  - realm provides login access
  - Samba provides resource sharing

- **Logging**
  - */var/log*
    - Primary directory for log storage on most Linux systems
    - Contains important logs such as
      - *syslog*
        - General system messages
      - *auth.log*
        - Authentication attempts
      - *dmesg*
        - Kernel ring buffer messages
    - Kernel ring buffer stores hardware, driver, and boot diagnostics
    - Commands for viewing logs
      - cat
      - less
      - tail
      - Example

        - tail -f /var/log/syslog to view real-time system messages
- *rsyslog*
    - Traditional logging service for Linux
    - Collects and routes messages from kernel, services, and apps
    - Categorizes logs by
        - Facilities (auth, daemon, mail, etc.)
        - Severities (debug, info, warning, critical, etc.)
    - Configuration files
        - /etc/rsyslog.conf
        - /etc/rsyslog.d/*.conf
    - Example rule
        - auth.info /var/log/auth.log to route auth facility messages at info level or higher
    - Service must be restarted after changes to apply configuration
- *journalctl*
    - Command-line tool for viewing logs from the systemd journal
    - Provides unified and filterable access to system logs
    - Aggregates kernel, service, and application logs
    - Commands
        - journalctl -b
            - View logs from current boot
        - journalctl -u ssh.service
            - View logs related to the SSH service
    - Output includes timestamps, services, and log messages
- *logrotate*
    - Manages log file rotation to prevent disk overuse

        - Compresses, renames, and deletes old logs on schedule
        - Configured via
            - /etc/logrotate.conf
            - /etc/logrotate.d/
        - Manual rotation example
            - sudo logrotate -f /etc/logrotate.conf
        - Rotated files renamed (e.g., syslog.1) and may be compressed (e.g., syslog.1.gz)
    - Summary
        - /var/log stores system, kernel, and authentication logs
        - rsyslog handles collection and routing of log data using facilities and severities
        - journalctl offers advanced filtering and unified viewing for systemd-based logs
        - logrotate keeps logs manageable by rotating and archiving them on schedule
        - Together, these tools support system monitoring, diagnostics, and stability

- **System Audit**
    - *auditd*
        - Background service for recording audit events to disk
        - Controlled with systemd using syntax
            - sudo systemctl <action> auditd
            - <action> can be start, stop, restart, enable, status

- Example command to check service and recent log output
  - sudo systemctl status auditd --no-pager --full
  - Displays service status, PID, log file path, and most recent events
- Example output detail
  - Service active
  - Process ID displayed
  - Audit log located at /var/log/audit/audit.log
  - Shows specific rules applied (e.g., monitoring /etc/passwd for wa events)
- Typical rule entry in logs includes
  - perms=wa
    - Logs write and attribute changes
  - key=identity
    - Custom label used to search logs

- *audit.rules*
  - Configuration file defining what the audit subsystem records
  - Located at /etc/audit/rules.d/audit.rules
  - Rules specify
    - File to monitor
    - Types of access to log
    - Label (key) for easy searching
  - Example rule
    - -w /etc/shadow -p wa -k passwd_changes
      - -w /etc/shadow
        - Watch target file
      - -p wa

- Permissions to monitor (write and attribute)
- -k passwd_changes
- Label for matching events
- After saving changes, reload or restart auditd to apply
- Events logged to /var/log/audit/audit.log
- Searchable using ausearch with the rule key
- Example
- ausearch -k passwd_changes

- Summary
- auditd records audit events to /var/log/audit/audit.log and is managed with systemctl
- audit.rules specifies which files or actions to monitor and tags them with user-defined keys
- Rules monitor files such as /etc/shadow and /etc/passwd for changes
- Keys act as searchable hashtags for locating relevant logs
- Tools like ausearch help filter logs by these keys for fast analysis and compliance

Firewall Implementation

Objective 3.2: *Given a scenario, configure and implement firewalls on a Linux system*

- **Firewalld Configuration and Management**
  - Overview
    - firewalld manages Linux firewall settings through zones and layered configuration
    - Zones define trust levels and allowed services per network interface
    - firewalld supports runtime and permanent rule sets
    - The firewall-cmd tool is used for querying and managing firewall settings
  - *Zones*
    - Zones are named trust profiles like public, home, work, internal, dmz, drop, and trusted
    - Each zone contains rules for allowed services and ports
    - Each network interface is bound to only one zone at a time
    - Services applied to a zone affect all interfaces assigned to that zone
    - New zones are created using
      - firewall-cmd --permanent --new-zone=<zone-name> && firewall-cmd --reload
    - View available zones using
      - firewall-cmd --get-zones
    - Example
      - DNS or HTTP services can be bound to a dmz zone for limited access from untrusted networks

- Runtime vs Permanent Settings
  - *Runtime settings*
    - Take effect immediately
    - Lost on reboot or reload
    - Ideal for testing and temporary adjustments
  - *Permanent settings*
    - Written to disk in /etc/firewalld/zones/
    - Survive reboots
    - Do not apply until a firewall-cmd --reload is executed
  - Allows for testing changes live before committing to disk
- *firewall-cmd* Usage
  - Command-line front-end for managing firewalld settings
  - Generic syntax
    - firewall-cmd [OPTIONS] [OPTION_VALUES]
  - Options include actions like --add-port, --add-service, --remove-port, etc.
  - Option values specify data like port number and protocol
  - Example
    - To allow HTTP traffic on the dmz zone permanently and apply it
      - firewall-cmd --zone=dmz --add-service=http --permanent && firewall-cmd --reload
  - Adds HTTP (port 80/tcp) to the dmz zone and reloads the firewall
- Common Options and Commands
  - --get-zones
    - Lists all available zones
  - --get-active-zones
    - Lists only zones with active interface bindings

        - --list-all --zone=<zone>
            - Displays all rules for a specific zone
        - --add-port=<port>/<proto>
            - Opens an individual port
        - --remove-port=<port>/<proto>
            - Closes an individual port
        - --runtime-to-permanent
            - Copies current runtime rules to the permanent configuration
        - --set-default-zone=<zone>
            - Changes the default zone for new interfaces
    - Summary
        - firewalld uses zones to organize traffic by trust level and manage allowed services
        - Each interface is assigned to one zone and inherits its rules
        - Runtime settings are immediate but temporary
        - Permanent settings are stored on disk and require a reload to activate
        - firewall-cmd is the essential tool to manage, query, and apply rules to runtime or permanent layers
        - A best practice is to test rules at runtime, save them permanently with --runtime-to-permanent, and reload to apply across reboots

- **Firewalld Rules and Access Control**
    - Overview
        - firewalld provides flexible access control by opening ports, enabling services, or using rich rules

- Services group multiple ports together under a name like https or ssh
- Rich Rules
  - Allow advanced matching with logging and conditional logic
  - Rules can be applied to a specific zone as either runtime (temporary) or permanent (persistent)

- Ports vs Services
  - Ports define individual protocol/port combinations such as 443/tcp
  - Services are predefined collections of ports under a name like https, dns, or ssh
  - Ports are opened using
    - firewall-cmd --zone=<zone> --add-port=<port>/<proto> [--permanent]
  - Services are enabled using
    - firewall-cmd --zone=<zone> --add-service=<service> [--permanent]
  - Example
    - Allow HTTPS traffic into the internal zone temporarily
      - firewall-cmd --zone=internal --add-service=https
        - --zone=internal scopes to the internal zone
        - --add-service=https enables TCP port 443
        - Absence of --permanent makes it runtime-only
- *Rich Rules*
  - Add complex conditional logic and logging capabilities
  - Generic syntax
    - firewall-cmd --zone=<zone> --add-rich-rule='<rule parts>' [--permanent]

- Rich Rules can filter traffic by
  - Source IP address or subnet
  - Network interface
  - Protocol or service
  - Logging behavior
  - Accept, reject, or drop action
- Example
  - Accept SSH from a VPN subnet and log each connection

```
firewall-cmd --zone=public \

--add-rich-rule='rule family="ipv4" source address="203.0.113.0/24" service name="ssh" log prefix="FW_SSH " level="info" accept' \

--permanent
```

  - --zone=public applies the rule to the public zone
  - family="ipv4" limits the rule to IPv4 traffic
  - source address="203.0.113.0/24" filters to the VPN subnet
  - service name="ssh" applies to TCP port 22
  - log prefix="FW_SSH " and level="info" enable syslog logging
  - accept is the final action
  - --permanent stores the rule in /etc/firewalld/zones/public.xml

- Summary
  - firewalld rules can be created using individual ports or predefined services

- Rules are applied to specific zones and can be either runtime or permanent
- Services like https enable all necessary ports automatically
- Rich Rules provide deeper control, allowing combinations of filtering, logging, and traffic decisions
- Administrators commonly test rules in runtime first, then commit them to permanent configuration using --permanent and firewall-cmd --reload

- **Uncomplicated Firewall**
  - *Uncomplicated Firewall (UFW)*
    - Is a simplified firewall front end maintained by the Ubuntu project
    - Designed to abstract iptables complexity with a plain-language syntax
    - Default behavior blocks all unsolicited inbound traffic and allows all outbound connections
    - Rule changes are immediate and persistent without requiring reloads or zone assignments
  - *Ports*
    - Ports can be opened directly by specifying port number and protocol
    - Syntax
      - sudo ufw allow <port>/<protocol>
    - Example
      - Allow web application traffic on TCP port 8080
        - sudo ufw allow 8080/tcp
          - sudo: invokes administrative privileges
          - ufw: is the tool

- allow: is the action
- 8080/tcp: is the socket being opened

- Results in an iptables rule applied immediately
- Simplifies the process compared to multi-line iptables or nftables definitions

- *Services*
  - UFW supports named application profiles stored in /etc/ufw/applications.d/
  - Profiles define one or more ports and protocols associated with common services
  - Example
    - Allow SSH access using predefined profile
      - sudo ufw allow OpenSSH
    - Profile label OpenSSH maps to 22/tcp
    - Profile Nginx Full may open both 80/tcp and 443/tcp
  - Advantages
    - Prevents port number typos
    - Improves readability and rule set documentation
- Persistent Configuration
  - UFW writes all rules directly to its configuration file
  - No distinction between runtime and permanent settings
  - Rules persist across reboots automatically
- Rule Management
  - Display current rules with line numbers
    - sudo ufw status numbered
  - Example output

- [1] 8080/tcp ALLOW Anywhere
- [2] OpenSSH ALLOW Anywhere

- Remove a rule by its index
  - sudo ufw delete 1
- Offers quick, index-based rule auditing and cleanup

- Summary
  - UFW simplifies Linux firewall management through concise syntax
  - Default settings block unsolicited inbound traffic and permit outbound communication
  - Ports can be managed directly by number and protocol or via named application profiles
  - Profiles are stored in /etc/ufw/applications.d/ and map to common service configurations
  - Rule changes are immediately effective and persistent across reboots
  - Commands like status numbered and delete streamline rule maintenance

- **Firewall Implementations**
  - Overview
    - iptables checks each packet individually against its rules and then admits or drops it
    - ipset groups large numbers of IP addresses or subnets into sets to allow faster lookups
    - nftables merges tables and rules into a unified, flexible, high-speed framework
  - *iptables*

- Classic Linux firewall framework
- Built around tables including filter, nat, mangle, raw, and security
- Each table contains chains such as INPUT, OUTPUT, and FORWARD
    - INPUT
        - Inspects packets destined for local services
    - OUTPUT
        - Filters packets created by local applications before sending
    - FORWARD
        - Filters packets routed through the system
- Rules are appended to chains to define what to do with matched traffic
- Generic syntax
    - iptables [-t <table>] -A <chain> -p <protocol> [match options] -j <target>
- iptables SSH example
    - sudo iptables -t filter -A INPUT -p tcp --dport 22 -j ACCEPT
        - -t filter
            - Selects the filtering table
        - -A INPUT
            - Appends rule to the INPUT chain
        - -p tcp
            - Matches TCP protocol
        - --dport 22
            - Matches destination port 22
        - -j ACCEPT
            - Allows the matched packet

- *ipset*

- High-performance address book for iptables to quickly match sets of IPs
- Syntax
  - ipset [options] <command> <setname> [params]
- Common commands include
  - create
  - add
  - del
  - list
- Common switches include
  - -exist
  - -!
- Used to create scalable IP blocklists
- ipset example
  - sudo ipset create blacklist hash:ip
  - sudo ipset add blacklist 203.0.113.5
  - sudo ipset list blacklist
  - ipset rule
    - Can be attached to iptables using -m set --match-set blacklist src -j DROP

- *nftables*
  - Modern replacement for iptables, ip6tables, ebtables, and arptables
  - Supports IPv4, IPv6, ARP, and Ethernet frame filtering
  - Uses transactional updates and simplified syntax
  - Syntax
    - nft [options] add rule <family> <table> <chain> <expression>
  - nftables SSH example

- sudo nft add rule inet filter input tcp dport 22 ct state new accept
    - inet matches both IPv4 and IPv6
    - filter is the table
    - input is the chain
    - tcp dport 22 matches destination port 22
    - ct state new limits to new connections
    - accept allows the packet

- Summary
    - iptables uses tables and chains to manage firewall rules and processes each rule sequentially
    - ipset enhances iptables by storing large IP lists for quick matching
    - nftables replaces all older firewall tools with a unified and flexible syntax supporting multiple protocols

- **Netfilter Module**
    - *Netfilter*
        - Built-in Linux kernel module for inspecting and controlling network traffic
        - Functions like a digital security checkpoint, evaluating every packet
        - Applies rules to determine whether packets are allowed or denied
        - Acts at the kernel level for fast and efficient filtering
    - Packet Filtering Framework
        - Defined set of rules that determine how packets are handled
        - Controls both incoming and outgoing traffic
        - Administrators configure rules to enforce security and traffic flow
    - iptables and nftables

        - Tools used by administrators to create and manage Netfilter rules
        - iptables is the traditional interface, still widely supported
        - nftables is the modern replacement, offering improved syntax and performance
        - Both tools define how packets are filtered, redirected, or logged
    - Kernel-Level Integration
        - Netfilter operates directly within the Linux kernel
        - Provides fast, low-overhead packet inspection and routing
        - Offers more flexibility and precision than third-party or external firewalls
        - Enables dynamic rule changes and deep packet inspection without added complexity
    - *firewalld*
        - Acts as a management front-end to simplify firewall rule configuration
        - Provides a user-friendly interface to manage zones and services
        - Still relies on Netfilter for actual packet filtering and enforcement
    - Summary
        - Netfilter is the kernel-level module that powers Linux firewall functionality
        - Uses a packet filtering framework defined by administrator rules
        - Tools like iptables and nftables manage these rules
        - Offers deep integration for high performance and granular control
        - firewalld simplifies configuration but uses Netfilter behind the scenes

- **Types of Address Translation**
    - *NAT (Network Address Translation)*

- Translates private internal IP addresses to a shared public IP address
- Enables multiple devices within a local network to access external networks using one public IP
- Commonly configured on routers or firewalls for IP conservation and added security
- Key terms
    - *Inside local*
        - Private IP address used within the local network (e.g. 192.168.10.20)
    - *Inside global*
        - Public IP address representing the internal host externally (e.g. 203.0.113.5)
- Example command for NAT configuration using SNAT
    - iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 203.0.113.5
        - -t nat
            - Applies to the NAT table
        - -A POSTROUTING
            - Appends the rule to the post-routing chain
        - -o eth0
            - Targets packets leaving through the eth0 interface
        - -j SNAT --to-source 203.0.113.5
            - Changes the source IP to the specified public address

- *PAT (Port Address Translation)*
    - Also called NAT overload

- Allows multiple internal devices to use a single public IP by assigning each connection a unique port number
- Tracks connections by IP and port combination, enabling simultaneous sessions through one IP
- Example
  - 192.168.10.20 → 203.0.113.5:51000
  - 192.168.10.30 → 203.0.113.5:51001
- Frequently used in home networks or small offices with limited IPs
- Example command for PAT configuration using MASQUERADE
  - iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
    - -t nat
      - Applies to the NAT table
    - -A POSTROUTING
      - Appends the rule after routing
    - -o eth0
      - Applies to outgoing traffic on interface eth0
    - -j MASQUERADE
      - Dynamically assigns source IP and port for each connection

- Summary
  - NAT translates internal private IPs into one public IP for outbound communication
  - PAT extends NAT by assigning unique ports to track each connection individually
  - NAT and PAT improve IP address efficiency and provide a layer of network security by hiding internal IP structures

- **Network Address Translation Variants**
  - *SNAT (Source Network Address Translation)*
    - Modifies the source IP address of packets as they leave the private network
    - Commonly used to translate private internal IPs (e.g. 192.168.10.50) into a single public IP (e.g. 203.0.113.5)
    - Ensures that return traffic from external servers is properly routed back to the originating internal host
    - Often used alongside PAT, modifying both IP addresses and port numbers
    - Example use case
      - Internal client sends a request to a web server
      - SNAT replaces 192.168.10.50 with 203.0.113.5 before it leaves the firewall
      - Web server replies to 203.0.113.5 and the firewall forwards it back to the original client
  - *DNAT (Destination Network Address Translation)*
    - Modifies the destination IP address of incoming packets before they reach internal systems
    - Commonly used to redirect public traffic (e.g. to a web server) to private internal IPs
    - Protects internal IP structure while still allowing access to specific services from outside the network
    - Example use case
      - Incoming HTTP request to 203.0.113.5:80

        - DNAT rewrites the destination to 192.168.10.100:80
        - Request reaches the internal web server without revealing its private IP externally
    - Summary
        - SNAT handles outbound traffic by rewriting source IP addresses to a public IP
        - DNAT handles inbound traffic by rewriting destination IP addresses to private internal IPs
        - Together, SNAT and DNAT allow private networks to communicate with the public internet securely and efficiently while preserving internal addressing schemes

- **Stateful vs Stateless**
    - Overview
        - Firewalls control inbound and outbound network traffic
        - Two types of firewalls in Linux
            - Stateless
            - Stateful
    - *Stateless Firewall*
        - Evaluates each packet independently without remembering previous traffic
        - Uses fixed rules to allow or block packets
        - Requires explicit rules for each allowed direction
        - Example rule set
            - iptables -A INPUT -p tcp --dport 80 -j ACCEPT

- iptables -A INPUT -j DROP
    - Accepts HTTP traffic (port 80), drops all else
- *Stateful Firewall*
    - Tracks and remembers active connections
    - Automatically allows responses to outgoing traffic
    - Simplifies rule management and improves security
    - Example rule set
        - iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
        - iptables -A OUTPUT -m state --state NEW,ESTABLISHED -j ACCEPT
        - iptables -A INPUT -j DROP
            - Allows outgoing connections and permits replies from those connections
- Summary
    - Stateless firewall checks each packet with no context, requiring full rule sets
    - Stateful firewall remembers connection states, allowing related traffic automatically
    - Stateful implementations are easier to manage and provide better security for active sessions

- **IP forwarding**
    - *IP Forwarding*
        - Enables a Linux system to pass IPv4 traffic between two network interfaces

- Allows the system to function as a router or gateway between different networks
- Useful when a device connects to both an internal network and the internet
- Prevents packet forwarding unless explicitly enabled

- *net.ipv4.ip_forward Setting*
  - Kernel parameter that controls IPv4 forwarding behavior
  - Located in the system's runtime configuration settings
  - Default value is 0 which disables IP forwarding
  - Setting it to 1 enables packet forwarding between interfaces
- *Temporary IP Forwarding*
  - Useful for testing or short-term use
  - Does not persist after reboot
  - Command syntax
    - sudo sysctl -w net.ipv4.ip_forward=1
  - This enables IP forwarding until the system is restarted
- *Permanent IP Forwarding*
  - Required for systems acting as routers or firewalls full-time
  - Changes must be made to a persistent config file
  - Add the following line to /etc/sysctl.conf
    - net.ipv4.ip_forward = 1
  - Apply the configuration with
    - sudo sysctl -p
- Summary
  - IP forwarding allows Linux to relay IPv4 traffic between networks
  - Controlled by net.ipv4.ip_forward, which defaults to disabled

- Use sysctl -w for temporary enabling
- Edit /etc/sysctl.conf for permanent configuration across reboots

Hardening Techniques

Objective 3.4: *Explain account hardening techniques and best practices*

- **Password Composition Controls**
  - Password Composition Overview
    - Controls enforce secure practices in user account management
    - They define rules for password creation to enhance system security
    - Most common controls include
      - Complexity
      - Length
    - Managed through the pam_pwquality module
    - Configured using the file /etc/security/pwquality.conf
  - *Password Complexity*
    - Ensures user passwords include a mix of character types
    - Character types include
      - Uppercase letters
      - Lowercase letters
      - Numbers
      - Special characters
    - Prevents creation of simple, easily guessed passwords
    - Controlled using the minclass option in /etc/security/pwquality.conf
    - Example configuration
      - Open the file with
        - sudo nano /etc/security/pwquality.conf

- Add or modify the line
  - minclass = 4
- Requires password to include characters from four different categories

- Enforced system-wide once saved
- Passwords not meeting the requirement are rejected upon creation or change

- *Password Length*
  - Sets the minimum number of characters required for a valid password
  - Longer passwords increase resistance to brute-force attacks
  - Controlled using the minlen option in /etc/security/pwquality.conf
  - Example configuration
    - Open the file with
      - sudo nano /etc/security/pwquality.conf
    - Add or modify the line
      - minlen = 12
    - Requires all passwords to be at least 12 characters long
  - Passwords shorter than the defined length are rejected
- Summary
  - Linux enforces password strength using composition controls managed by pam_pwquality
  - Configuration is performed in /etc/security/pwquality.conf
  - minclass defines required variety of character types
  - minlen sets the minimum acceptable password length
  - Together these controls help ensure strong, secure passwords across the system

- **Password Lifecyle Controls**
    - Overview
        - Password lifecycle controls manage how long passwords are valid and how they are used over time
        - They promote security by enforcing expiration, preventing reuse, and maintaining uniqueness
        - Three components of password lifecycle management are
            - Expiration
            - History
            - Reuse
    - *Expiration*
        - Expiration sets how long a password can be used before requiring a change
        - Managed using the chage command
        - Syntax
            - sudo chage -M [max_days] [username]
        - Example
            - sudo chage -M 90 jdoe
                - Sets jdoe's password to expire every 90 days
                    - -M option defines the maximum age in days
    - *History*
        - Password history tracks a user's previously used passwords
        - Enables enforcement of reuse restrictions
        - Works with reuse policies to enforce uniqueness

- *Reuse*
  - Prevents immediate reuse of previous passwords
  - Enforced using the pam_pwhistory module
  - Configured in /etc/pam.d/common-password
  - Configuration line
    - password required pam_pwhistory.so remember=5
      - remember=5
        - Stores last 5 passwords and blocks reuse
  - Error message when reused
    - Password has been already used. Choose another.
- Summary
  - Expiration defines the valid period for a password
  - History records recent passwords for each user
  - Reuse blocks the user from reapplying recent passwords
  - Implemented with tools like chage and pam_pwhistory
  - Ensures password changes are meaningful and secure over time

- **MFA**
  - Overview
    - Multi-Factor Authentication (MFA) adds multiple identity checks for enhanced security
    - MFA in Linux is implemented using
      - PAM configurations
      - OTP solutions
      - Hardware-token integration

- *PAM Configurations*
  - PAM stands for Pluggable Authentication Modules
  - Configuration files are located in /etc/pam.d
  - PAM separates authentication into four interfaces
    - auth
    - account
    - password
    - session
  - Control flags determine the rule's behavior in the authentication stack
    - required
      - Failure is noted but does not halt processing
    - requisite
      - Failure causes immediate denial
    - sufficient
      - Success grants access if no required failures exist
    - optional
      - Only relevant if no other modules of the same type run
  - Generic syntax
    - type control_flag module_path module_options
  - Example command to append a rule for UNIX password check
    - echo 'auth required pam_unix.so nullok_secure' | sudo tee -a /etc/pam.d/sshd
  - This command
    - Adds an authentication rule
    - Uses required flag
    - References pam_unix.so module

        - Allows empty passwords only on secure terminals using nullok_secure
    - *OTP Solutions*
        - One-Time Password (OTP) solutions add a software-based second factor
        - Google Authenticator is a common module
        - Generic syntax
            - auth control_flag pam_google_authenticator.so [options]
        - Example command
            - echo 'auth required pam_google_authenticator.so nullok' | sudo tee -a /etc/pam.d/sshd
        - This command
            - Requires time-based code
            - Uses nullok to allow login if OTP not yet set up
    - *Hardware-token Integration*
        - Physical keys like YubiKey are supported using pam_yubico
        - Generic syntax
            - auth control_flag pam_yubico.so id=<client_id> authfile=<path> [options]
        - Example command
            - echo 'auth required pam_yubico.so id=16 authfile=/etc/yubico/authorized_yubikeys debug' | sudo tee -a /etc/pam.d/sshd
        - This command
            - Requires physical YubiKey for login
            - Uses id=16 for client ID
            - Reads allowed tokens from specified file

        - Enables verbose logging with debug
    - Summary
        - MFA strengthens security by layering multiple authentication methods
        - PAM provides a flexible framework with four interfaces and control flags
        - OTP solutions integrate via PAM modules like pam_google_authenticator.so
        - Hardware tokens like YubiKey require additional PAM configuration using pam_yubico
        - Each method reinforces user identity verification during login

- **Checking existing breach lists**
    - Overview
        - Personal or organizational data may appear in breach lists unknowingly
        - Breach-checking tools help identify exposure
        - Three main tools covered
            - haveibeenpwned
            - DeHashed
            - Intelx.io
    - *haveibeenpwned.com*
        - Used to check if email addresses have appeared in known breaches
        - Accessible via web at https://haveibeenpwned.com
        - Can be used with a free API key
        - Example API call
            - https://haveibeenpwned.com/api/v3/breachedaccount/alice@example.com

- Returns a JSON array listing breach names
- Ideal for
  - Quick audits
  - Integration into cron-based Bash scripts
- Focuses only on email breaches
- Known for
  - Speed
  - Reliability
  - Simplicity
- Does not support searching by phone numbers, IPs, or document dumps

- *DeHashed*
  - Expands breach intelligence beyond email
  - Searches include
    - Usernames
    - Email addresses
    - Phone numbers
    - IP addresses
    - Leaked documents
  - Requires an API key
  - Outputs JSON or CSV results
  - Supports advanced features (with paid tier)
    - Filtering by date, type, sensitivity
    - Email alerts
    - Bulk export
  - Good for
    - More contextual insights

        - Monitoring multiple accounts
    - Provides deeper results than HIBP
    - Requires more setup effort
- *Intelx.io*
    - Provides enterprise-grade breach intelligence
    - Aggregates data from
        - Public dumps
        - Dark-web forums
        - Paste sites
        - Obscure online sources
    - Used via RESTful API
    - Outputs richly tagged data
    - Supports integration with SIEM tools
        - Elasticsearch
        - Splunk
        - Custom Python scripts
    - Key features
        - Regex, wildcard, proximity queries
        - Deep archived data
    - Best suited for
        - Large-scale investigations
        - Vulnerability management pipelines
    - Has a steep learning curve and licensing cost
    - Offers unmatched visibility and query power
- Summary
    - haveibeenpwned

        - Free
        - Email-only breach checks
        - Simple and fast
    - DeHashed
        - Paid features
        - Multi-vector search including phone, IP, documents
        - Alerts and export options
    - Intelx.io
        - Enterprise-focused
        - Deep breach analysis
        - Advanced query syntax and SIEM integration

- **Restricted shell use**
    - */sbin/nologin*
        - Used to prevent interactive login for a user
        - Commonly applied to service or system accounts not intended for human interaction
        - Displays a message and denies access if login is attempted
        - Allows account to own files and execute scheduled tasks without allowing shell access
        - Example command
            - sudo useradd -s /sbin/nologin backupbot
                - backupbot is created as a user who cannot log in interactively but can still perform background operations
    - */bin/rbash*

        - Restricted version of the Bash shell
        - Permits login but limits user actions
        - Prevents changing directories, setting environment variables, or executing arbitrary programs
        - Useful for junior staff or external users who require minimal shell access
        - Example command
            - sudo useradd -s /bin/rbash -m reports
                - reports user is created with restricted access and a home directory
    - Summary
        - /sbin/nologin: Completely disables shell access for service accounts while allowing file ownership and automation
        - /bin/rbash: Enables controlled shell access with restrictions on commands and environment modifications
        - Both tools are used to limit user capabilities based on their role and enhance overall system security

- **pam_tally2**
    - Configuration
        - PAM (Pluggable Authentication Modules) defines how authentication is handled in Linux
        - *pam_tally2*
            - Is a PAM module used to add login tracking and lockout functionality
        - Configuration is added to files such as

- /etc/pam.d/common-auth
- /etc/pam.d/login

- Example configuration line
  - auth required pam_tally2.so onerr=fail deny=5 unlock_time=600
    - Locks the account after 5 failed attempts
    - Automatically unlocks it after 600 seconds (10 minutes)

- Failed login tracking
  - pam_tally2 logs failed login attempts per user account
  - Useful in identifying brute-force attacks or user confusion
  - Command to check login failure status for a specific user
    - pam_tally2 --user jsmith
  - Displays number of failed attempts and allows administrators to take action
- Account lockouts
  - Enforced after reaching the failure threshold defined in configuration
  - Blocks user login until unlock_time expires or admin resets count
  - Helps prevent unauthorized access from repeated failed login attempts
  - Lockout behavior strengthens enterprise system security
- Summary
  - pam_tally2 is a PAM module that monitors failed login attempts and triggers lockouts
  - Configured by editing PAM authentication files with rules like deny and unlock_time
  - Tracks login failures on a per-user basis and supports administrative review with pam_tally2

        - Enforces account lockouts to protect against unauthorized access and brute-force attacks

- **Avoid running as root**
    - Overview
        - Root access provides unrestricted system control but introduces security risks
        - Linux alternatives for administrative tasks include sudoers and polkit
    - *sudoers*
        - Grants specific users permission to run defined commands with elevated privileges
        - Managed through the /etc/sudoers file
        - Must be edited using the visudo command to prevent syntax errors
        - Example entry
            - jsmith ALL=(ALL) /sbin/systemctl restart apache2
                - jsmith
                    - User the rule applies to
                - First ALL
                    - Applies on any host
                - (ALL)
                    - Run command as any user (typically root)
                - /sbin/systemctl restart apache2
                    - Exact allowed command
        - Enables least privilege access by limiting command scope
    - *polkit (PolicyKit)*

- Framework for managing authorization in graphical and background services
- Tools included
  - pkexec
    - Run commands as another user
  - pkaction
    - List privileged actions
  - pkcheck
    - Verify user authorization for actions
  - pkttyagent
    - Provide authentication prompts in terminal sessions
- Enables fine-grained control over privileged actions in modern environments

- Summary
  - Running as root increases risk of unintended system damage
  - sudoers allows users to execute only specified commands as root
  - polkit supports graphical and service-level authorization using a policy-based model
  - Tools like pkexec, pkcheck, and pkaction help enforce the principle of least privilege
  - Both systems support secure delegation of administrative tasks without full root access

Cryptographic Concepts

Objective 3.5: *Explain cryptographic concepts and technologies in a Linux environment*

- **Data at Rest - File Encryption**
  - *GPG (GNU Privacy Guard)*
    - Data at rest encryption protects files stored on disk from unauthorized access
    - Even if a system is compromised or files are copied, encryption keeps data unreadable without the correct key
    - GPG is commonly used for file encryption in Linux environments
    - GPG supports encryption, decryption, and digital signing using asymmetric cryptography
  - *Asymmetric encryption*
    - Uses two keys
      - Public key to encrypt
      - Private key to decrypt
    - Allows secure sharing of sensitive files, like configuration data or credentials
    - Only the person with the matching private key can read the encrypted file
    - Ensures confidentiality, even if files are intercepted or copied
  - Command-line usage
    - gpg
      - Is the command-line tool used to encrypt, decrypt, and sign files
    - Comes pre-installed or available through the package manager

- Encrypting a file
  - gpg --encrypt --recipient 'bob@example.com' confidential.txt
  - Encrypts confidential.txt so only bob@example.com (with private key) can decrypt
- Decrypting a file
  - gpg --decrypt confidential.txt.gpg
  - Prompts user for their private key's passphrase to access content
- Digital signatures
  - GPG can also be used to digitally sign files
  - Signing verifies the sender's identity and ensures file integrity
  - Useful in enterprise environments for verifying patches, scripts, or updates
  - Helps prevent impersonation and tampering in distributed systems
- Summary
  - Data at rest encryption ensures stored files remain secure and unreadable without proper keys
  - GPG offers encryption, decryption, and digital signing using asymmetric public key cryptography
  - Administrators use GPG to share files securely and verify file authenticity
  - gpg tool provides command-line access to encrypt, decrypt, and sign files with ease
  - Incorporating GPG improves trust, security, and compliance across Linux systems

- **Data at Rest - Filesystem Encryption**
    - *LUKS2 (Linux Unified Key Setup version 2)*
        - LUKS2 is a standardized on-disk encryption container used for full filesystem encryption
        - It provides encryption through the cryptsetup command-line tool
        - Protects data at rest by encrypting entire disk partitions or volumes
        - Ensures unauthorized users cannot read stored files even with physical access
    - Key cryptsetup commands
        - *cryptsetup isLuks*
            - Checks if a device contains a LUKS header
        - *cryptsetup luksFormat*
            - Initializes a new encrypted container
        - *cryptsetup luksOpen*
            - Maps an encrypted device to /dev/mapper
        - *cryptsetup luksClose*
            - Unmaps the encrypted device
        - *cryptsetup luksAddKey*
            - Adds a new passphrase or keyfile
        - *cryptsetup luksRemoveKey*
            - Removes an existing passphrase
    - LUKS2 container creation
        - Syntax
            - cryptsetup luksFormat [options] <device>
        - Example
            - sudo cryptsetup luksFormat --type luks2 /dev/sdb1

- Formats /dev/sdb1 with LUKS2 encryption and prompts for a passphrase
- All management commands operate after initialization

- *Argon2 (Key-Derivation Function)*
  - Integrated directly into LUKS2 container headers via cryptsetup luksFormat
  - Memory-hard function slows down brute-force attacks by requiring memory and CPU time
  - No external config files
    - Parameters are stored in the LUKS2 header
- Argon2 configuration parameters
  - *Memory cost*
    - Amount of RAM required (e.g., 64 MiB or more)
  - *Time cost*
    - Number of iterations over the memory (commonly 2 or 3)
  - *Parallelism*
    - Number of CPU threads used simultaneously
- Argon2 variants
  - *argon2i*
    - Protects against side-channel attacks
  - *argon2d*
    - Resists GPU-based brute-force attacks
  - *argon2id*
    - Hybrid of i and d, balancing protections
- Inspection
  - Use cryptsetup luksDump to inspect a container's Argon2 settings
- Summary

- LUKS2 provides a full-featured on-disk encryption solution with management via cryptsetup
- Argon2 ensures strong key derivation with resistance to brute-force and side-channel attacks
- Parameters for memory, time, and parallelism are specified during formatting
- Together, LUKS2 and Argon2 create a hardened, configurable framework for secure file storage on Linux

- **Data in Transit - OpenSSL**
  - Overview
    - OpenSSL secures data in transit using certificates and encryption
    - Key components are certificate and key management and the s_client diagnostic tool
  - *Certificate and Key Management*
    - TLS certificates function like digital passports for servers
    - Certificates contain the server's public key and identifying information
    - Certificates are signed by a trusted Certificate Authority (CA)
    - Private keys are securely generated and stored on the server
    - Generate a Private Key
      - Command
        - openssl genpkey -algorithm RSA -out server.key
      - With encryption
      - openssl genpkey -algorithm RSA -out server.key -aes256 -pkeyopt rsa_keygen_bits:2048

- Create a Certificate Signing Request (CSR)
  - Command
    - openssl req -new -key server.key -out server.csr
      - CSR includes the public key and identification details
      - Submitted to a CA for certificate signing
- Verify the CSR
  - Command
    - openssl req -in server.csr -noout -text
  - Displays subject information, public key, and extensions
- Certificate Lifecycle Management
  - Administrators track expiration dates, renew, or revoke certificates
  - Enterprise environments may automate key storage, CSR generation, and certificate deployment
  - Common tools include HSMs and vaults for secure key storage

- *s_client*
  - openssl s_client probes TLS-enabled services from the command line
  - Command example
    - openssl s_client -connect mail.example.com:993 -showcerts
      - Retrieves server certificate chain and verifies details
  - Additional Options
    - -servername
      - Tests Server Name Indication (SNI)
    - -cipher
      - Forces use of a specific cipher suite
    - Helps detect misconfigurations such as

        - Missing intermediate certificates
        - Expired certificates
        - Disabled or unsupported protocol versions
    - Summary
        - OpenSSL enables TLS communication by managing certificates and keys
        - Administrators create keys and CSRs, then install signed certificates
        - openssl s_client is used to test certificate chains and TLS connections
        - Proper use of OpenSSL ensures encrypted, authenticated communication over networks

- **Data in Transit Protection Methods**
    - *TLS Protocol Version*
        - TLS (Transport Layer Security) is the foundational protocol for encrypted internet communication
        - TLS 1.2 and TLS 1.3 are currently considered secure
        - TLS 1.2+ supports authenticated encryption algorithms and forward secrecy
        - Forward secrecy ensures each session uses a unique key that cannot be reused
        - The TLS handshake involves ClientHello and ServerHello messages, followed by key exchange and session setup
        - Earlier versions like TLS 1.0 and 1.1 are deprecated due to weak cipher support and vulnerability to downgrade and padding oracle attacks
    - *LibreSSL*
        - LibreSSL is a security-focused fork of the OpenSSL library
        - Designed to be smaller, easier to audit, and safer for production use

- Removes outdated features and legacy code to reduce attack surface
- Installed using
  - sudo apt install libressl
- Maintains compatibility with familiar OpenSSL commands
- Example
  - openssl genpkey -algorithm RSA -out server.key
  - openssl req -new -key server.key -out server.csr

- *WireGuard*
  - WireGuard is a modern VPN implemented inside the Linux kernel
  - Encrypts entire network tunnels with modern cryptography
  - Uses simple, peer-based configuration files
  - Delivers high throughput and low latency ideal for remote work or site-to-site VPNs
  - Easily integrated with orchestration and monitoring tools
  - Enterprises use it for secure, fast, and scalable VPN deployment
  - Supports secure key storage via vaults or hardware security modules (HSMs)
- Summary
  - TLS 1.2 and 1.3 protect data in transit with modern encryption and session isolation
  - LibreSSL is a hardened replacement for OpenSSL that removes legacy vulnerabilities and simplifies management
  - WireGuard provides kernel-level VPN security with high performance and easy configuration for scalable enterprise use

- **Hashing**
    - *SHA-256*
        - SHA-256 is a cryptographic hash algorithm that produces a 256-bit (64-character) digest
        - Hashing ensures integrity by changing completely with even a single character alteration in the input
        - Hash values are used to verify that file contents have not been altered or corrupted
        - The Linux utility sha256sum computes the SHA-256 hash of a file
        - Command example
        - sha256sum document.txt
        - Output is a hexadecimal hash followed by the filename
    - *HMAC (Hash-based Message Authentication Code)*
        - HMAC combines a cryptographic hash function (like SHA-256) with a shared secret key
        - Provides both data integrity and authenticity
        - The OpenSSL tool can generate an HMAC using the dgst command
        - Command example
            - openssl dgst -sha256 -hmac "mysecretkey" document.txt
                - The resulting HMAC confirms the file is unchanged and was generated by someone with the same secret key
    - Summary
        - SHA-256 ensures data integrity by producing a unique digest for each input
        - The sha256sum command verifies that downloaded files remain unchanged

        - HMAC adds a shared secret to the hashing process, ensuring both integrity and authenticity
        - OpenSSL's dgst command with -hmac enables HMAC generation
        - Hash comparison protects against corruption and tampering, especially when downloading software

- **Removal of Weak Algorithms**
    - *DISA STIGs (Defense Information Systems Agency Security Technical Implementation Guides)*
        - DISA STIGs define hardening baselines for Linux servers in government and defense environments
        - STIGs explicitly require the removal or disabling of weak cryptographic algorithms
        - Examples include
            - Disabling legacy ciphers such as RC4 and 3DES
            - Prohibiting the use of MD5-based hashing algorithms
        - Enforcing strong TLS protocol versions across services
    - *FIPS (Federal Information Processing Standards) 140-2*
        - FIPS 140-2 defines the cryptographic modules and algorithms approved for federal systems
        - FIPS compliance ensures only approved algorithms are used on Linux systems
        - Eliminates weak algorithms such as MD5 and RC4
        - Approved primitives include
            - AES in approved modes

- SHA-2
- RSA
- ECDSA

- Applies to OpenSSL, OpenSSH, and VPN configurations

- *Disabling SSHv1*
  - SSH version 1 is deprecated and contains known cryptographic vulnerabilities
  - Disabling SSHv1 is a key hardening step to enforce secure connections using SSHv2
  - Command example

```
sudo sed -i 's/^#\?Protocol .*/Protocol 2/' /etc/ssh/sshd_config
sudo systemctl restart sshd
```

  - The sed command updates the SSH configuration to force Protocol 2
  - Restarting sshd applies the changes and removes fallback to SSHv1
- *sslscan*
  - sslscan is a command-line tool for auditing enabled cryptographic algorithms on TLS services
  - It reports supported protocol versions and cipher suites
  - Identifies weak configurations including
    - TLS 1.0
    - RC4
    - MD5-based MACs
  - Used to confirm only secure algorithms remain active after hardening
- Summary
  - Weak cryptographic algorithms must be removed to protect systems from modern attacks

- DISA STIGs define hardening rules including the removal of outdated ciphers and insecure hashes
- FIPS 140-2 ensures the use of only federally approved algorithms in critical systems
- Disabling SSHv1 enforces the exclusive use of SSHv2 with stronger encryption
- sslscan provides verification by scanning TLS services and identifying insecure algorithms

- **Certificate Management**
  - *No-cost Trusted Root Certificates*
    - No-cost certificates are provided by free public Certificate Authorities or created as self-signed
    - *Let's Encrypt*
      - Is a free, publicly trusted CA that issues domain-validated certificates
      - Trusted by all major browsers and platforms
      - Suitable for blogs, personal projects, and small-scale services
      - Automatically issues and renews certificates
    - *Self-signed certificates*
      - Are created independently without a third-party CA
      - Free and fully controlled by the administrator
      - Not trusted by clients unless manually added to their trust stores
      - Causes browser warnings by default
      - Best for internal development, testing, or isolated networks

- Cost Trusted Root Certificates
  - *Commercial root CAs*
    - Issue certificates for enterprise and regulated use cases
  - Examples of commercial CAs
    - DigiCert
    - GlobalSign
    - Sectigo
    - Entrust
  - Requires generation of a private key and Certificate Signing Request (CSR)
  - CSR is submitted to a Registration Authority (RA) for verification
  - After approval, the CA issues a signed certificate
  - Features of paid certificates
    - Extended validation (EV)
    - Long validity periods
    - Insurance warranties
    - Dedicated technical support
  - Used for customer-facing websites, enterprise services, and secure environments requiring high assurance
- Summary
  - Certificate management ensures the authenticity and trust of digital services
  - No-cost certificates include Let's Encrypt and self-signed options
    - Let's Encrypt is trusted and automated for public use
    - Self-signed certificates are for internal or controlled environments
  - Cost-based certificates come from commercial CAs with extensive validation and premium support

        - Paid options are better suited for regulated industries, legal assurance, and brand protection
        - The choice between no-cost and paid certificates depends on the desired balance of trust, assurance, cost, and complexity

- **Self-Signed Certificates**
    - Self-Signed Certificate Use
        - *Self-signed certificates*
            - Are generated and signed with the user's own private key
            - Suitable for internal, non-public environments where trust can be controlled
            - Appropriate for
                - Development servers
                - Internal staging environments
                - Isolated corporate services
            - Example command to generate a self-signed certificate
                - openssl req -x509 -newkey rsa:2048 -keyout dev.key -out dev.crt -days 365 -nodes -subj "/CN=dev.example.local"
            - Certificate and key can be used to configure HTTPS for internal web servers
                - dev.crt for certificate
                - dev.key for private key
            - Certificate must be manually distributed or deployed through configuration management tools to trusted systems

- Expiration must be tracked and certificates regenerated as needed per policy

- *Self-Signed Certificate Avoidance*
  - Not suitable for public-facing services such as e-commerce or client portals
  - Lacks a chain of trust to a public root Certification Authority
  - Triggers browser warnings like "Not Secure" that reduce user confidence
  - Increases risk of on-path (man-in-the-middle) attacks due to unverifiable identity
  - Undermines trust models when users are trained to bypass security warnings
  - Trusted root-signed certificates should be used for services accessed by external users
- Summary
  - Self-signed certificates allow internal systems to authenticate without third-party involvement
  - Proper for use in environments where trust distribution is controlled and external validation is unnecessary
  - Improper for public or user-facing services where third-party validation and browser trust are required
  - Trusted authority-signed certificates should be used when authenticity and security assurance are essential for users and clients

Compliance and Audit Procedures

Objective 3.6: *Explain the importance of compliance and audit procedures*

- **Detection and Response**
  - Anti-malware
    - *ClamAV*
      - Open-source anti-malware for Linux
      - Scans filesystem against malware signature database
      - Example
        - sudo clamscan -r /usr
    - *Linux Malware Detect (LMD)*
      - Builds on ClamAV
      - Targets malware in web-hosting environments
      - Scans uploads for PHP backdoors and known threats
    - *rkhunter*
      - Detects rootkits and suspicious modifications
      - Example
        - sudo rkhunter --check
    - *chkrootkit*
      - Scans for hidden binaries, altered libraries, and rootkits
  - *Indicators of Compromise (IoCs)*
    - Clues left behind by attackers
      - Unexpected processes
      - Unauthorized file changes

        - Abnormal network connections
    - *grep*
        - Searches logs for brute-force login attempts
        - Example
            - grep -i 'failed password' /var/log/auth.log
    - *ss*
        - Shows open ports and listening services
        - Example
            - ss -tulnp
    - *YARA*
        - Custom rule-based scanner for malware detection
    - *auditd and Wazuh*
        - Host-based IDS tools
        - Monitor critical files and alert on changes
- Summary
    - Anti-malware tools like ClamAV, LMD, rkhunter, and chkrootkit detect malicious files and behaviors
    - IoCs include log anomalies, unknown processes, and unauthorized network access
    - Tools such as grep, ss, YARA, auditd, and Wazuh help identify and respond to threats
    - Combined, anti-malware and IoC tools support proactive detection and system integrity monitoring

- **Vulnerability Categorization**
    - *CVE (Common Vulnerabilities and Exposures)*
        - CVE is a unique identifier assigned to publicly disclosed software flaws
        - Managed by MITRE at cve.mitre.org in collaboration with CVE Numbering Authorities
        - Published in the U.S. National Vulnerability Database (NVD) at nvd.nist.gov
        - Format is CVE-YYYY-NNNNN
            - YYYY represents the year of assignment
            - NNNNN is the unique sequence number
        - CVEs allow consistent reference to specific vulnerabilities across teams and tools
        - CVEs include descriptions, links to vendor advisories, patches, and are accessible via public APIs
    - *CVSS (Common Vulnerability Scoring System)*
        - CVSS assigns a severity score to each CVE ranging from 0.0 to 10.0
        - Scores reflect factors such as
            - Exploit complexity
            - Required privileges
            - Impact on confidentiality, integrity, and availability
        - Maintained by FIRST at first.org/cvss
        - Three metric groups determine the final score
            - Base
            - Temporal
            - Environmental

        - Scores appear in the National Vulnerability Database and can be integrated into vulnerability management tools
        - CVSS score ratings
            - None: 0.0
            - Low: 0.1–3.9
            - Medium: 4.0–6.9
            - High: 7.0–8.9
            - Critical: 9.0–10.0
    - Summary
        - Vulnerability categorization provides a standardized method for identifying and describing software flaws
        - CVE assigns unique identifiers to vulnerabilities, allowing consistent reference and tracking
        - CVSS provides numeric severity scores for each vulnerability, helping prioritize based on risk
        - Together, CVE and CVSS create a unified framework for vulnerability identification and remediation prioritization across security teams

- **Vulnerability Management**
    - *Service Misconfigurations*
        - Occur when Linux daemons are configured with unsafe defaults or overly permissive settings
        - Example
            - SSH configured to allow password-based root logins
            - Allows brute-force attacks to gain root access

- Mitigated by editing /etc/ssh/sshd_config to disable PermitRootLogin and requiring key-based authentication
- Restart SSH service after changes

- Another example
  - Binding critical services to all network interfaces (0.0.0.0)
  - Exposes services like databases or admin panels to the external Internet
  - Mitigated by setting the listen address to 127.0.0.1 or a private subnet
  - Enforce with firewall rules

- *Backporting Patches*
  - Applies security fixes from newer versions to older distributions
  - Avoids full system upgrades while still patching vulnerabilities
  - Used in long-lifecycle enterprise environments prioritizing stability and compatibility
  - Distribution maintainers adapt official patches to the older codebase
  - Rebuilt packages are published to the distribution's security or backports repository
  - Administrators subscribe to the repository and use standard update tools to install
  - Updates should be tested in staging before being deployed
- Summary
  - Vulnerability management includes correcting misconfigurations and applying security updates

- Misconfigurations such as unsafe SSH settings or improperly bound services must be addressed with proper configuration and network controls
- Backporting patches allows enterprises to secure older systems without performing disruptive full upgrades
- Together, these practices ensure Linux systems remain both secure and stable

- **Vulnerability Detection**
  - *Port Scanners*
    - Used to detect open TCP and UDP ports and identify associated services
    - Help administrators find misconfigurations and potential attack surfaces
    - Open ports are compared to windows in a house
      - Necessary for function but dangerous if left unsecured
    - *Nmap*
      - Open-source industry-standard port scanner
      - Capable of ping sweeps, OS fingerprinting, and version detection
      - Example
        - nmap -sS -sV 10.0.0.0/24
        - Performs a stealth SYN scan and service version detection
        - Identifies unauthorized or outdated services
    - *Zenmap*
      - Graphical interface for Nmap
      - Suitable for junior admins
      - Allows saving scan profiles and mapping network topologies

        - Useful for documentation and audits
    - *Protocol Analyzers*
        - Allow deep inspection of data flowing through network connections
        - Used to understand service behavior, verify configuration, and detect threats
        - *Wireshark*
            - Graphical packet analysis tool
            - Decodes hundreds of protocols
            - Example
                - Filter http.request used to inspect HTTP payloads for exposed credentials
        - *Tshark*
            - Command-line version of Wireshark
            - Suitable for scripts and automation
            - Ideal for regular log reviews and diagnostics
        - *tcpdump*
            - CLI packet sniffer used in server and remote environments
            - Example
                - sudo tcpdump -i eth0 port 443 -w capture.pcap
                - Captures HTTPS traffic to a file for offline analysis
    - Summary
        - Vulnerability detection involves identifying weaknesses before exploitation
        - Port scanners like Nmap and Zenmap detect open ports and services
        - Protocol analyzers like Wireshark, Tshark, and tcpdump inspect network traffic

        - These tools help verify configurations, detect threats, troubleshoot performance, and support forensic analysis

- **Standards and Audit**
    - Overview
        - Securing IT systems requires both structured standards and verification tools
        - CIS Benchmarks provide security configuration guidelines
        - OpenSCAP automates auditing and compliance checking
    - *CIS Benchmarks*
        - Developed by the Center for Internet Security (CIS)
        - Serve as expert-reviewed checklists to secure system configurations
        - Cover multiple platforms including Linux, Windows, and cloud services
        - Used to
            - Harden systems against vulnerabilities
            - Ensure consistent security across many machines
            - Meet industry compliance requirements
        - Access information
            - Freely available from cisecurity.org
            - Requires creating a free account to download
        - Examples of guidance
            - Password policy configuration
            - Disabling unused services
            - Setting file permissions
        - Suitable for

        - Single server hardening
        - Large-scale enterprise audit preparation
    - *OpenSCAP*
        - Stands for Open Security Content Automation Protocol
        - Open-source tool using SCAP standards for automated compliance checking
        - Automates the following tasks
            - Scanning systems for compliance with security benchmarks
            - Identifying configuration gaps
            - Generating audit reports
            - Suggesting or applying corrective fixes
        - Integrates with benchmarks like CIS
        - Allows scheduling of regular scans and tracking changes over time
        - Supports enterprise-scale auditing without manual checks
        - Improves consistency and security posture across systems
    - Summary
        - CIS Benchmarks provide structured security guidance for system configuration
        - OpenSCAP enables automated compliance checking against such standards
        - Together, they support secure, efficient, and auditable system administration

- **File Integrity Verification**
    - Overview

- File integrity verification ensures software and system files remain authentic and unchanged
- Protects against tampering, corruption, and unauthorized modifications
- Involves two major processes
  - Signed package verification
  - Installed file verification

- *Signed Package Verification*
  - Confirms authenticity and integrity of software packages before installation
  - Acts like a digital seal to prove a package came from a trusted source
  - Tools used
    - apt for Debian-based systems
    - dnf for Red Hat-based systems
  - Verifies package signatures using cryptographic methods
  - Ensures software has not been altered in transit
  - Example
    - Verifying Ubuntu updates were signed by Canonical before applying them
  - Automatically performed by modern Linux package managers
  - Critical for secure software deployment across many machines in enterprise settings
- *Installed File Verification*
  - Used after installation to ensure files remain unchanged
  - Detects unauthorized modifications or file corruption
  - Tools used:
    - rpm -V for Red Hat-based systems

- debsums for Debian-based systems
- Compares file attributes like
  - Size
  - Permissions
  - Cryptographic hashes
- Example
  - Verifying the integrity of the sshd binary if a server behaves unexpectedly
- Allows fast detection of tampering or accidental damage
- Helps IT teams respond quickly to potential threats

- Summary
  - Signed package verification checks package authenticity before installation
  - Installed file verification ensures system files remain intact after installation
  - Together, they provide a complete system for maintaining file security and integrity on Linux systems

- **File Integrity Tools**
  - Overview
    - File integrity tools help detect unauthorized or unexpected changes to system files
    - Useful for preventing or identifying corruption, misconfigurations, or malicious activity
    - Two common tools are

        - rkhunter
        - AIDE
    - *rkhunter (Rootkit Hunter)*
        - Scans system for known rootkits, backdoors, and vulnerabilities
        - Compares files and settings against a threat-signature database
        - Suitable for quick health checks
        - Key Commands
            - rkhunter --check
                - Prompts between scan sections
            - rkhunter --check --skip-keypress
                - Runs full scan without pause
            - rkhunter --update
                - Updates the local threat-signature database
        - Scan sections may include
            - Hidden file checks
            - Rootkit signature scan
            - System command verification
            - File permission review
        - Can be scheduled with cron to run daily and notify administrators via email
        - Lightweight and beginner-friendly
        - Provides detection for known threats and basic system anomalies
    - *AIDE (Advanced Intrusion Detection Environment)*
        - Performs in-depth file integrity monitoring by capturing file metadata
        - Builds a custom baseline and compares files to detect changes over time
        - Key Commands

- aide --init
  - Creates initial database of file attributes
- aide --check
  - Compares current system state to baseline

- File attributes captured include
  - Permissions
  - Ownership
  - Size
  - Modification time
  - Cryptographic checksums
- Commonly scheduled with cron or systemd timers for regular scans
- Baseline should be saved in a secure, read-only location
- Results can be logged or fed into SIEM tools like Logwatch
- Ideal for compliance-sensitive environments such as
  - Healthcare
  - Finance
  - Government
- Offers comprehensive coverage of system changes beyond threat signatures
- Requires more configuration than rkhunter but provides greater control

- Summary
  - rkhunter detects known malware by checking for rootkit signatures and system anomalies
  - AIDE builds a baseline snapshot and verifies file changes over time for deep integrity analysis

        - Both tools contribute to secure system maintenance and proactive threat detection in enterprise Linux environments

- **Data Destruction Overwriting Tools**
    - *Data Destruction Overwriting Tools*
        - Overwrite existing data to prevent recovery
        - Useful for securely deleting files or wiping full storage devices
        - Commonly used in sensitive environments before decommissioning drives
        - Three key tools
            - shred
            - dd if=/dev/urandom
            - badblocks -w
    - *shred*
        - Used to securely delete files by overwriting with random data
        - Prevents file recovery after deletion
        - Syntax
            - shred [options] filename
        - Example
            - shred -u -v -n 3 secrets.txt
                - -n 3
                    - Three overwrite passes
                - -v
                    - Verbose output
                - -u

- Remove file after overwriting
- Output
  - shred: secrets.txt: pass 1/3 (random)

    shred: secrets.txt: pass 2/3 (random)

    shred: secrets.txt: pass 3/3 (random)

    shred: secrets.txt: removed
    - Each pass is listed
    - Final message confirms file removal
- Effective for user-level file destruction

- *dd if=/dev/urandom*
  - Used to overwrite entire devices or partitions with random data
  - Suitable for full disk or partition wiping
  - Syntax
    - dd if=/dev/urandom of=/dev/device bs=blocksize
  - Example
    - dd if=/dev/urandom of=/dev/sdb bs=1M status=progress
      - bs=1M
        - 1MB blocks
      - status=progress
        - Shows progress in real time
  - Output
    - Displays total bytes written and write speed
  - Effectively destroys data across entire storage medium
- *badblocks -w*
  - Typically used to identify bad disk sectors
  - In write-mode, overwrites disk with patterns to wipe data

- Syntax
  - badblocks -wsv /dev/device
- Example
  - badblocks -wsv /dev/sdb
    - -w
      - Write mode
    - -s
      - Show status
    - -v
      - Verbose
- Writes and verifies patterns like 0xAA, 0x55, 0xFF
- Output
  - Checking for bad blocks in read-write mode

    From block 0 to 625142447

    Testing with pattern 0xaa: done

    Reading and comparing: done

    Testing with pattern 0x55: done
    - Indicates blocks tested and verification of each pattern
- Slower but provides disk health testing in addition to data wipe

- Summary
  - shred
    - Overwrites and deletes individual files
  - dd if=/dev/urandom
    - Overwrites entire disks or partitions with random bits
  - badblocks -w
    - Overwrites entire disk with patterns and checks disk health

        - All tools help prevent recovery of sensitive data and ensure secure decommissioning of storage media

- **Cryptographic Data destruction**
    - Overview
        - Cryptographic data destruction makes encrypted data permanently unreadable by destroying the encryption key
        - Unlike traditional methods like shred or dd, this approach does not require overwriting all data
        - Two common tools used in Linux for this process are cryptsetup with LUKS and zuluCrypt
    - *cryptsetup with LUKS*
        - Encrypts entire disk partitions in Linux
        - Destruction method involves erasing the LUKS header or one or more keyslots
        - LUKS header stores keyslots and encryption metadata
        - Erasing header makes encrypted data permanently inaccessible without overwriting the data
        - Command syntax
            - cryptsetup luksErase /dev/device
        - Example
            - cryptsetup luksErase /dev/sdb1
                - Erases LUKS header on /dev/sdb1
                - This operation is irreversible
    - *zuluCrypt*

- Front-end interface for cryptsetup with both GUI and CLI options
- Simplifies management and destruction of encrypted volumes
- Syntax
  - zuluCrypt-cli -z -d /dev/device
- Example
  - zuluCrypt-cli -z -d /dev/sdb1
    - -z
      - Indicates LUKS header erasure
    - -d
      - Specifies the device
- Offers prompts and warnings to reduce risk of accidental data loss
- Suitable for users who prefer guided operations

- Summary
  - Cryptographic destruction removes access by deleting encryption keys instead of data
  - cryptsetup luksErase offers a fast, secure, command-line solution
  - zuluCrypt-cli provides a safer, user-guided interface for the same purpose
  - Effective in enterprise environments where security and efficiency are critical

- **Software Supply Chain**
  - Overview
    - The software supply chain is the ecosystem behind software development
    - Includes code, tools, libraries, contributors, and processes

- Security risks include tampering, untrusted sources, and hidden vulnerabilities
- Three key concepts for securing the supply chain
  - GPG
  - SBOM
  - CI/CD

- *GPG (GNU Privacy Guard)*
  - Ensures software comes from a trusted source
  - Verifies integrity and authenticity of code and packages
  - Developers use GPG to sign
    - Code commits
    - Software releases
  - Recipients verify signatures using the developer's public key
  - Protects against tampered or spoofed packages
  - Automatically integrated into CI/CD pipelines for verification
  - Common in enterprise environments for secure code distribution
- *SBOM (Software Bill of Materials)*
  - SBOM list of all components in a software project
  - Identifies libraries, dependencies, and third-party modules
  - Enhances transparency and vulnerability tracking
  - Enables quick response when a vulnerability is discovered in a dependency
  - Automatically generated using tools during development
  - Required for
    - Compliance
    - Security audits

        - Incident response
    - Important in enterprise settings for risk management
    - Tool knowledge not required for exam
- *CI/CD (Continuous Integration / Continuous Deployment)*
    - CI/CD automates build, test, and release workflows
    - Keeps development fast, consistent, and secure
    - Ensures each code change follows a predefined, secure process
    - Steps in pipeline often include
        - Automated testing
        - GPG signature validation
        - SBOM generation and analysis
    - Builds are rejected if
        - Code is unsigned
        - Vulnerable dependencies are found
    - Enforces secure practices throughout the development lifecycle
- Summary
    - The software supply chain must be secured to maintain trust and prevent tampering
    - GPG
        - Validates software origin and integrity
    - SBOM
        - Provides a full inventory of software components
    - CI/CD
        - Automates secure delivery through enforced policies and tests
    - Together, these methods help secure modern software from development to deployment

- **Security Banners**
    - Overview
        - Security banners provide messages to users before or after login
        - Used for system identification, warnings, policy messages, and announcements
        - Especially important in multi-user or enterprise systems
        - Key files used
            - /etc/issue
                - Shown before login on local terminals
            - /etc/issue.net
                - Shown before login over remote access like SSH
            - /etc/motd
                - Shown after successful login
    - */etc/issue*
        - Displays a pre-login message on local terminals
        - Common use
            - Show system identity or warning
        - Example message
            - Authorized access only. This is Server A - Production.
    - */etc/issue.net*
        - Displays a pre-login message to remote users (SSH, Telnet)
        - Common use
            - Legal or policy warnings
        - Example message

        - Warning: Unauthorized access to this system is prohibited and will be prosecuted.
    - */etc/motd*
        - Stands for Message of the Day
        - Displays a post-login message
        - Used for announcements, system status, or reminders
        - Example message
            - System maintenance scheduled for Saturday at 10 PM. Please save your work.
    - Summary
        - /etc/issue
            - Local terminal pre-login message
        - /etc/issue.net
            - Remote pre-login message
        - /etc/motd
            - Post-login message
        - Used for system awareness, legal notices, and real-time updates to users

Operating System Hardening

Objective 3.3: *Given a scenario, apply operating system (OS) hardening techniques on a Linux system*

- **sudo Configuration**
  - Overview
    - sudo stands for superuser do
    - Allows regular users to execute commands with elevated privileges
    - sudo configuration is managed through controlled files and editing tools
    - Three key components of sudo configuration
      - visudo
      - /etc/sudoers
      - /etc/sudoers.d
  - *visudo*
    - Safest method for editing sudo configuration
    - Opens configuration in vi and checks syntax before saving
    - Prevents mistakes that could lock out administrative access
    - Common options
      - -c
        - Checks syntax without editing
      - -f /path/to/file
        - Edits a specific file, e.g., in /etc/sudoers.d
      - -s
        - Enables strict mode
      - -x

        - Exports sudoers content in JSON format
    - Supports managing access through user groups like wheel
    - Example
        - usermod -aG wheel alice
            - Grants sudo privileges to user alice
    - Used in enterprise environments to standardize sudo management
- */etc/sudoers*
    - Main configuration file for defining sudo rules
    - Accessed and modified using visudo
    - Highly secured and should never be edited directly
    - Defines
        - Which users can run what commands
        - Under which conditions
        - Global sudo policies
    - Example entries
        - admin ALL=(ALL) ALL grants full privileges to admin
        - jane ALL=(ALL) /usr/bin/systemctl restart apache2 restricts jane to restarting Apache only
    - Supports advanced settings
        - Timeout durations for password prompts
        - Environment variable control
        - Command aliases for grouping command rules
    - Central file for managing structured and flexible sudo access
- */etc/sudoers.d*
    - Directory for modular and segmented sudo configuration
    - Enables per-team or per-role sudo rules

- Supports better policy management and easier automation
- Example files
  - web-team file for web-related sudo rules
  - dev-ops file for broader operational permissions
- Must be edited using visudo -f /etc/sudoers.d/<filename>
- Enhances auditability and minimizes risks when updating access policies

- Summary
  - sudo controls who can execute commands with root privileges
  - visudo ensures safe and validated sudo configuration editing
  - /etc/sudoers contains global sudo rules and advanced access settings
  - /etc/sudoers.d enables modular configuration using separate rule files
  - Together, these tools support secure, flexible, and organized administration of privileged access on Linux systems

- **sudoers Directives**
  - Overview
    - sudoers file controls not only who can run commands but how they behave
    - Directives enhance control and fine-tune access behavior
    - Two commonly used directives
      - NOPASSWD
      - NOEXEC
  - *NOPASSWD*
    - Allows users to execute specific sudo commands without password prompt

- Useful in automation scenarios or frequent, low-risk administrative actions
- Reduces user friction but may decrease accountability and increase security risk
- Syntax
  - username ALL=(ALL) NOPASSWD: /path/to/command
- Example
  - jane ALL=(ALL) NOPASSWD: /usr/bin/systemctl restart apache2
- Command behavior
  - Jane can restart Apache using sudo without a password
  - Command
    - sudo /usr/bin/systemctl restart apache2
- Security note
  - Should be used with care to avoid privilege misuse in production environments

- *NOEXEC*
  - Prevents sudo-permitted commands from launching additional programs or subshells
  - Enhances security by stopping escalation through embedded shells
  - Best used when allowing commands like text editors or pagers
  - Syntax
    - username ALL=(ALL) NOEXEC: /path/to/command
  - Example
    - alex ALL=(ALL) NOEXEC: /usr/bin/less /var/log/syslog
  - Command behavior
    - Alex can use sudo to view logs but cannot open a shell inside less

- Command
  - sudo /usr/bin/less /var/log/syslog
- Attempting to use ! inside less will fail

■ Security note

- Limits abuse of interactive commands with embedded shell capabilities

○ Summary

■ sudoers directives manage both access and behavior of privileged commands

■ NOPASSWD removes password prompts for specific commands

■ NOEXEC blocks command-level privilege escalation via subprocesses

■ These directives help enforce tight security policies in Linux environments while enabling flexible workflows for trusted users

- **sudo User Groups**
  - Overview
    - Administrative access is commonly managed using user groups
    - sudo and wheel groups are used to assign sudo privileges
    - Users in these groups can run commands with elevated (root) privileges
    - Command sudo -i opens a full administrative shell for group members
  - *sudo Group*
    - Default sudo group on Debian-based systems (e.g., Ubuntu)
    - Grants administrative privileges to group members
      - Example command to add a user to sudo group
        - sudo usermod -aG sudo alice

- alice must log out and back in to gain privileges
- Allows execution of commands like
- sudo apt update
- First sudo use prompts for user's password
- [sudo] password for alice:
- Setup used to delegate controlled admin access in Ubuntu environments
- *wheel Group*
- Default group for sudo access on Red Hat-based systems (e.g., RHEL, Fedora)
- Historical naming convention
- Example command to add a user to wheel group
- sudo usermod -aG wheel bob
- bob logs out and back in to activate group permissions
- Can use sudo to run commands after password prompt
- [sudo] password for bob:
- Useful for group-level privilege management in enterprise systems
- *sudo -i*
- Command to open full root shell for users with sudo privileges
- Simulates a login as the root user
- Removes need to prefix each command with sudo
- User prompted for their password before elevation
- Prompt changes to indicate root shell
- Session remains elevated until exit is entered
- Depends on entries in sudoers file
- %sudo ALL=(ALL:ALL) ALL
- %wheel ALL=(ALL:ALL) ALL

        - Should be used with caution due to potential system-wide impact
    - Summary
        - sudo group is used for sudo access on Debian-based systems
        - wheel group is used for sudo access on Red Hat-based systems
        - Group-based privilege management simplifies access control
        - Commands
        - sudo usermod -aG sudo <username>
        - sudo usermod -aG wheel <username>
        - sudo -i gives unrestricted administrative access via root shell
        - Proper sudoers file configuration required for functionality
        - Understanding these groups and commands is essential for secure and efficient Linux administration

- **Root shell**
    - Overview
        - The root shell provides complete administrative control over the system
        - sudo su - is used to access the root shell with full environment settings
        - Root shell access allows performance of system-wide changes and maintenance
        - Commands run as root affect the entire system and must be used carefully
    - *sudo su -*
        - sudo
            - Is a command used to execute tasks with elevated privileges
        - su

- Stands for substitute user
- -
  - Loads the full environment of the target user
- sudo su -
  - Combines these to switch to the root shell with full root environment
- Grants access to root's environment variables and configuration
- Requires an entry in the sudoers file like
  - <username> ALL=(ALL:ALL) ALL
- Prompts for the user's own password, not the root password

- Example Use Case
  - Administrator wants to update system software packages
  - Runs
    - sudo su -
      - Enters their own password when prompted
      - Gains root shell access
  - Then runs
    - apt update && apt upgrade -y
      - This updates and upgrades all installed packages with elevated permissions
- Summary
  - Root shell grants full control over a Linux system
  - sudo su - allows switching to root with full environmental inheritance
  - User must have sudo privileges set in sudoers file
  - System prompts for user's password, not the root's
  - Commonly used for tasks requiring complete administrative access

        - Misuse or mistakes in root shell can have major consequences

- **File Attributes**
    - Overview
        - File attributes provide control beyond standard file permissions
        - Two commands used for managing file attributes:
            - lsattr
            - chattr
    - *lsattr*
        - Displays current attributes of files or directories
        - Syntax
            - lsattr [options] [files]
        - Common options
            - -R lists attributes recursively
            - -a includes hidden files
            - -d shows directory attributes instead of their contents
            - -v shows version number if supported
        - Example
            - lsattr -a -d -R /etc/myconfig
                - Shows attributes in /etc/myconfig, its subdirectories, and hidden files
        - Sample output
            - ----i--------e-- /etc/myconfig/settings.conf (immutable)
            - -----a--------e-- /etc/myconfig/logs.log (append-only)
    - *chattr*

- Changes file or directory attributes
- Syntax
    - chattr [options] [attributes] [files]
- Common options
    - -R applies changes recursively
    - -v works with file versioning
- Attribute flags
    - +i sets file as immutable
    - -i removes immutability
- Examples
    - chattr +i /usr/local/bin/myscript.sh
        - Makes file read-only, even to root
    - chattr -i /usr/local/bin/myscript.sh
        - Removes immutability
    - chattr -R +i /usr/local/bin/scripts/
        - Protects all files in a directory recursively

- Summary
    - lsattr shows which file attributes are active
    - chattr adds or removes attributes
    - Immutability protects files from changes, deletions, or renaming—even by root
    - Useful for securing important system files beyond regular permissions

- **File Permisssions**
    - Overview

- Every file and directory in Linux has a set of permissions
- Permissions are defined for
  - File owner
  - Assigned group
  - All other users
- File access is managed using
  - chown to change file owner and optionally group
  - chgrp to change only the group ownership

- *chown* Command
  - Used to change ownership of a file or directory
  - Syntax
    - chown [options] new_owner[:new_group] file
  - Options
    - -R
      - Apply changes recursively to all files and subdirectories
    - -v
      - Display verbose output of changes
  - Example
    - To assign ownership of /data/reports to user jane and group analytics recursively and verbosely
      - sudo chown -Rv jane:analytics /data/reports
  - Sample Output
    - changed ownership of '/data/reports/summary.csv' from root:root to jane:analytics
    - changed ownership of '/data/reports/2024/quarter1.xlsx' from root:root to jane:analytics

- *chgrp* Command
  - Used to change only the group ownership of a file or directory
  - Syntax
    - chgrp [options] new_group file
  - Options
    - -R
      - Apply group changes recursively
    - -v
      - Display verbose output of changes
  - Example
    - To assign group developers to /project/files recursively and verbosely
      - sudo chgrp -Rv developers /project/files
- Summary
  - File permissions in Linux determine read, write, and execute access
  - Permissions apply to owner, group, and others
  - Ownership is controlled using
    - chown to change owner and optionally group
    - chgrp to change group only
  - Recursive and verbose flags help manage and confirm bulk changes efficiently

- **File Modifications**
    - Overview
        - File permissions control who can read, write, or execute files and directories
        - Each file has permissions for three user types
            - User/Owner
            - Group
            - Others
        - The chmod command modifies these permissions
        - Two modes of chmod usage
            - Symbolic notation
            - Octal notation
    - *chmod with Symbolic Notation*
        - Syntax
            - chmod [who][operator][permission] <file>
                - [who] options
                    - u (user/owner)
                    - g (group)
                    - o (others)
                    - a (all)
                - [operator] options
                    - + (add permission)
                    - - (remove permission)
                    - = (set exact permission)
                - [permission] options
                    - r (read)

        - w (write)
        - x (execute)
    - Example command
        - chmod u+x deploy.sh
            - Adds execute permission to user/owner for deploy.sh
    - Example permission change
        - Before
            - -rw-r--r--
        - After
            - -rwxr--r–
- *chmod with Octal Notation*
    - Syntax
        - chmod [mode] <file>
    - Each permission has a value
        - read = 4
        - write = 2
        - execute = 1
    - Each digit sets permission for one category: user, group, others
    - Example command
        - chmod 754 server_config.txt
            - User
                - 7 (r+w+x)
            - Group
                - 5 (r+x)
            - Others
                - 4 (x)

- Resulting permissions
    - -rwxr-xr-- 1 root root 2048 May 24 10:15 server_config.txt
- Summary
    - chmod changes file and directory permissions
    - Symbolic mode allows granular adjustments
    - Octal mode applies all permissions in one step
    - Useful for ensuring secure and appropriate access control

- **File Special Permissions**
    - Overview
        - Special permissions provide advanced access control beyond standard read, write, execute bits
        - Apply to files and directories for security and consistency
        - Three main types
            - setuid
                - Run file with owner's privileges
            - setgid
                - Enforce group ownership on execution or new files
            - sticky bit
                - Restrict file deletion in shared directories
    - *setuid*
        - Grants users the ability to run a program with the permissions of the file's owner
        - Commonly used to allow normal users to perform privileged tasks
        - Syntax

- chmod u+s <file>
- Example
  - To allow any user to run changeconfig with root privileges
    - sudo chown root:root changeconfig && sudo chmod u+s changeconfig
- ls Output
  - Shows an s in the user execute position
    - -rwsr-xr-x 1 root root 12345 May 24 10:00 changeconfig

- *setgid*
  - Applies group ownership persistently to files or directories
  - On files: runs with the file's group privileges
  - On directories: new items inherit the directory's group
  - Syntax
    - chmod g+s <file_or_directory>
  - Example
    - To enforce devteam group ownership in /shared/projects
      - sudo chgrp devteam /shared/projects && sudo chmod g+s /shared/projects
  - ls Output
    - Shows s in the group execute field
      - drwxr-sr-x 2 root devteam 4096 May 24 10:05 /shared/projects
- *sticky bit*
  - Prevents users from deleting files they don’t own in shared directories
  - Useful in public writable directories
  - Syntax

        - chmod +t <directory>
    - Example
        - To restrict deletion in /public
            - sudo chmod +t /public
    - ls Output
        - Shows t at the end of the permission string
            - drwxrwxrwt 7 root root 4096 May 24 10:10 /public
- Summary
    - setuid
        - Run as file owner
    - setgid
        - Inherit group ownership or run with group privileges
    - sticky bit
        - Protect shared files from deletion by others
    - Displayed in ls -l as s or t in execute bits
    - Essential for secure multi-user and shared Linux environments

- **Default File Creation Mask**
    - Overview
        - Linux applies default permissions to newly created files and directories
        - *umask (user file creation mask)*
            - Defines which permission bits are removed from system defaults
            - Allows control over default file security and accessibility
    - umask Functionality
        - Controls default permissions by subtracting from system defaults

- Default modes
  - Files
    - 666 (read/write for user, group, others)
  - Directories
    - 777 (read/write/execute for all)
- No execute bit for files by default for security
- umask removes bits from these base permissions

- Common umask Values
  - umask 022
    - Removes write for group and others
    - File permissions
      - 644 (rw-r--r--)
    - Directory permissions
      - 755 (rwxr-xr-x)
  - umask 027
    - Removes write for group and all access for others
    - File permissions
      - 640 (rw-r-----)
    - Directory permissions
      - 750 (rwxr-x---)
- Checking and Setting umask
  - Check current umask
    - $ umask
      - Output
        - 0022 (leading 0 represents no special bits being masked)

        - Set umask in shell session
            - umask 022
        - Make permanent by adding to shell config file
            - ~/.bashrc
            - ~/.profile
    - Summary
        - umask masks bits from default 666 (files) or 777 (directories)
        - Defines default permission behavior for all new files/directories
        - Common setting 022 creates files with 644 and directories with 755
        - Stricter setting 027 creates files with 640 and directories with 750
        - Adjusting umask helps enforce security policies across user environments

- **ACLs**
    - Overview
        - Standard Linux permissions support one owner, one group, and others
        - ACLs (Access Control Lists) extend permissions beyond this limitation
        - Allow individual users or groups to have specific access rights to files/directories
        - Useful in shared environments requiring fine-grained access control
    - *getfacl*
        - Views current ACL entries on files or directories
        - Syntax
            - getfacl [options] <file_or_directory>
        - Common option
            - -R to view recursively

- Example
  - getfacl -R project/
- Sample output

```
# file: project/report.txt
# owner: admin
# group: admin
user::rw-
user:alice:r--
group::r--
mask::r--
other::---
```

  - Displays standard and ACL-specific entries
  - The mask sets maximum effective permissions for named users and groups

- *setfacl*
  - Sets, modifies, or removes ACL entries
  - Syntax
    - setfacl [options] <permissions> <file_or_directory>
  - Common options
    - -m
      - To modify or add
    - -x
      - To remove
    - -R
      - To apply recursively
  - Example
    - setfacl -m u:bob:rw report.txt
      - Grants user bob read and write access
      - Afterward, getfacl report.txt would show

        - user:bob:rw-
    - Summary
        - ACLs extend Linux permissions to assign rights to multiple users or groups
        - getfacl views current ACL settings, including extended entries
        - setfacl modifies or removes ACL entries without altering ownership
        - ACLs are layered on top of traditional permissions
        - Provide flexibility in managing complex file access scenarios

- **SELinux States**
    - Overview
        - SELinux acts as a security guard enforcing access control rules
        - It operates in three states
            - Disabled
            - Permissive
            - Enforcing
        - States determine how strictly SELinux enforces policies on the system
    - *Disabled*
        - SELinux is completely turned off
        - No policy enforcement occurs
        - No policy violations are logged
        - Typically used for troubleshooting or non-SELinux environments
        - Not recommended for long-term use due to lack of protection
        - To permanently disable
            - Edit /etc/selinux/config
            - Set SELINUX=disabled

- Reboot required for change to take effect
- To check state
  - getenforce
- Output
  - Disabled

- *Permissive*
  - SELinux is active but does not enforce rules
  - Logs actions that would have been denied
  - Useful for testing and debugging
  - Allows system operations to continue without interruption
  - To set temporarily
    - setenforce 0
  - To set permanently
    - Edit /etc/selinux/config
    - Set SELINUX=permissive
- *Enforcing*
  - SELinux actively enforces policies and blocks unauthorized actions
  - Ideal for secure, production environments
  - Ensures integrity and controlled access across the system
  - To enable temporarily
    - setenforce 1
  - To set permanently
    - Edit /etc/selinux/config
    - Set SELINUX=enforcing
- Summary
  - Disabled

        - SELinux is off
        - No control or logging
    - Permissive
        - SELinux logs but does not block actions
    - Enforcing
        - SELinux actively blocks unauthorized actions
    - Temporary state changes use setenforce
    - Persistent configuration requires editing /etc/selinux/config

- **SELinux Mode Management**
    - Overview
        - SELinux adds an extra layer of access control in Linux by enforcing security policies
        - Modes include Enforcing, Permissive, and Disabled
        - Administrators use commands to check and modify SELinux behavior during configuration and troubleshooting
        - Two key commands
            - getenforce
            - setenforce
    - *getenforce*
        - Used to check the current SELinux mode
        - Syntax
            - getenforce
        - Does not take any options or arguments
        - Returns one of the following states

- Enforcing
- Permissive
- Disabled

- Example
  - Run
    - getenforce
  - Output
    - Enforcing
- Non-intrusive and safe to run anytime
- Commonly used to verify mode before and after applying configuration changes

- *setenforce*
  - Used to temporarily change SELinux mode at runtime
  - Only supports switching between Enforcing and Permissive modes
  - Cannot enable SELinux from Disabled mode
  - Requires reboot and config file change to enable SELinux if disabled
  - Syntax
    - setenforce [0|1]
      - 1 sets Enforcing mode
      - 0 sets Permissive mode
  - Example
    - Run sudo setenforce 0 to switch to Permissive
    - Takes effect immediately without reboot
  - Useful for temporary debugging or maintenance
- Summary
  - getenforce shows current SELinux mode

- setenforce switches between Enforcing and Permissive temporarily
- Changes from Disabled state require editing /etc/selinux/config and reboot
- Together, these commands provide runtime control over SELinux behavior

- **SELinux File Security Contexts**
  - Overview
    - SELinux assigns a security context label to every file and directory
    - These labels define access rules enforced by SELinux policy
    - Incorrect or missing contexts can block access even with correct Linux permissions
    - Tools for managing contexts
      - ls -Z
      - restorecon
      - chcon
  - *ls -Z*
    - Displays SELinux security context of files and directories
    - Output includes
      - SELinux user (e.g., system_u, unconfined_u)
      - Role (e.g., object_r)
      - Type (e.g., httpd_sys_content_t)
      - Level (e.g., s0)
    - Syntax
      - ls -Z <path>
    - Example

- ls -Z /var/www/html
- Sample output

```
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 index.html
```

- Type field is most critical for SELinux policy access decisions
- Level field is often set to s0 in single-level systems

- *restorecon*
  - Resets SELinux context to default policy-defined value
  - Useful for fixing mislabeling due to file movement or incorrect creation
  - Syntax
    - restorecon [options] <path>
  - Common options
    - -R for recursive
    - -v for verbose
  - Example
    - sudo restorecon -Rv /var/www/html
      - Restores all files under the path to their correct SELinux context
- *chcon*
  - Manually sets SELinux context components on a file or directory
  - Syntax
    - chcon [options] <file>
  - Options include
    - -u for SELinux user
    - -r for role

- -t for type
- Example

```
sudo chcon –u system_u –r object_r –t httpd_sys_content_t index.html
```

- Makes context changes visible via ls -Z
- Manual changes made with chcon may be overridden by restorecon or system policy tools

- Summary
  - SELinux file contexts consist of user, role, type, and level
  - ls -Z displays current context
  - restorecon resets to default context from policy
  - chcon manually sets custom contexts
  - Correct context labeling is essential for SELinux access control

- **SELinux system-wide configuration**
  - Overview
    - SELinux controls system-wide access using configurable toggles called booleans
    - Booleans act like on/off switches to adjust how strict or flexible SELinux policies behave
    - Three main tools used
      - getsebool
      - setsebool
      - semanage

- Booleans can control service behavior such as HTTPD access, Samba, or FTP home directory access

- *getsebool*
  - Displays current status (on/off) of SELinux booleans
  - Syntax
    - getsebool [boolean_name]
    - getsebool -a
  - Example
    - getsebool httpd_enable_homedirs
  - Output
    - httpd_enable_homedirs --> off
  - Other common booleans
    - ftp_home_dir
      - Allows FTP access to user home directories
    - httpd_can_network_connect
      - Allows HTTPD to initiate outbound connections
    - samba_enable_home_dirs
      - Allows Samba to share user home directories
- *setsebool*
  - Changes SELinux boolean values (on/off)
  - Temporary change
    - setsebool boolean_name on|off
  - Persistent change
    - setsebool -P boolean_name on|off
  - Example
    - setsebool -P httpd_enable_homedirs on

        - -P flag ensures the change survives reboots
        - No output is shown unless an error occurs
    - *semanage*
        - Used to persistently modify SELinux policy settings
        - Syntax for boolean
            - semanage boolean -m --on|--off boolean_name
        - Example
            - semanage boolean -m --on httpd_enable_homedirs
        - Always applies persistent changes to SELinux policy
        - Also used for
            - Port contexts
                - Allow services to use non-default ports
            - File labels
                - Define how SELinux treats files/directories, especially in custom locations
    - Summary
        - getsebool checks boolean state
        - setsebool sets boolean temporarily or permanently
        - semanage provides permanent SELinux policy changes
        - Booleans adjust specific security behaviors, while port contexts and file labels extend control to network and file-based resources

- **SELinux Logging and Troubleshooting**
    - Overview
        - SELinux enforces strict access policies and logs violations quietly

- Troubleshooting SELinux issues requires reading and interpreting audit logs
- Two key tools assist in this process
  - sealert
    - Explains what was denied and why
  - audit2allow
    - Helps generate policy rules to allow denied behaviors

- *sealert*
  - Analyzes SELinux audit logs and provides human-readable explanations
  - Reads from /var/log/audit/audit.log
  - Syntax
    - sealert -a /path/to/audit.log
  - Example
    - sealert -a /var/log/audit/audit.log
  - Sample output

```
SELinux is preventing /usr/sbin/httpd from name_connect access on the tcp_socket port
```

  - Provides solution suggestions, such as
    - setsebool -P httpd_can_network_connect 1
      - Helps identify the cause of access denials and appropriate corrective actions

- *audit2allow*
  - Generates SELinux policy module suggestions from audit logs
  - Converts denial logs into readable allow rules

- Syntax to analyze current audit log
  - audit2allow -a
- Syntax to specify input log file
  - audit2allow -i /path/to/log
- Example output
  - #============= httpd_t ==============

    allow httpd_t mysqld_port_t:tcp_socket name_connect;
    - Used to create custom SELinux policy modules
    - Must be used cautiously to avoid weakening security by allowing unsafe behaviors

- Summary
  - SELinux logs all denied actions for review
  - sealert translates log entries into readable reports with recommendations
  - audit2allow helps build policy modules to safely permit necessary access
  - Always confirm safety of any changes before applying new policy rules

- **SSHD Secure Authentication and Access Control**
  - Overview
    - SSHD is the entry point for remote login to a Linux system
    - Controls both authentication method and access permissions
    - Key features include key vs password authentication, PermitRootLogin, AllowUsers, and AllowGroups
    - Configuration managed via /etc/ssh/sshd_config
  - Key vs Password Authentication
    - Two authentication methods available

- Password-based
  - Users enter a password during login
- Key-based
  - Uses a public-private key pair for authentication

- Key-based is more secure and resistant to brute-force attacks
- To enforce key-based login

```
PasswordAuthentication no
systemctl restart sshd
```

- Disables login attempts using passwords only

- *PermitRootLogin*
  - Controls whether the root account can log in remotely via SSH
  - Root logins are risky due to full system privileges
  - To disable root SSH login

```
PermitRootLogin no
systemctl restart sshd
```

- Encourages use of non-root accounts with sudo for privilege escalation

- *AllowUsers*
  - Specifies which individual user accounts may log in via SSH
  - Useful for environments with restricted access requirements
  - Format
    - AllowUsers alice bob

        - Can include user@hostname for precise control
        - Add to /etc/ssh/sshd_config and restart SSH service to apply
    - *AllowGroups*
        - Grants SSH access based on Linux group membership
        - Useful for managing access for roles or teams
        - Format
            - AllowGroups sshusers
                - Only users in the specified group may log in via SSH
                - Easier to manage access by changing group memberships
    - Summary
        - SSHD manages secure access to Linux via remote login
        - Key-based authentication is more secure than password-based
        - Root logins should be disabled with PermitRootLogin no
        - Use AllowUsers to permit specific user accounts
        - Use AllowGroups to grant access to all members of a designated group

- **SSHD Secure Configuration and Usage**
    - Overview
        - SSHD manages remote access with encrypted and authenticated connections
        - Default settings may include unnecessary or risky features
        - Secure configuration improves system protection and reliability
        - Two key focus areas
            - Disabling X11 forwarding
            - Understanding SSH tunneling

- *Disabling X11 Forwarding*
  - X11 forwarding allows remote graphical apps to display locally
  - Rarely needed on servers
  - Introduces potential attack vectors if left enabled
  - Disable X11 forwarding by editing the SSHD config file
    - X11Forwarding no
  - File location
    - /etc/ssh/sshd_config
  - Restart SSH service after changes
    - systemctl restart sshd
  - Disabling this feature is a simple way to harden SSH
- *SSH Tunneling*
  - SSH tunneling routes traffic securely through an SSH connection
  - Commonly used to access internal services securely
  - Tunnels encrypt data for protocols that aren't secure by default
  - Example use
    - Forward a port for a private internal web server
  - SSHD supports tunneling by default—no config change needed
  - Administrators should audit and monitor tunneling in sensitive environments
  - Misuse can lead to bypassing firewall or access control restrictions
- Summary
  - SSHD is central to secure remote administration
  - Disabling unnecessary features like X11 forwarding reduces risk
  - SSH tunneling is powerful but must be monitored for potential misuse

        - Secure configuration of SSHD helps maintain a strong system security posture

- **Secure Remote Access**
    - Overview
        - SSH provides encrypted remote access to Linux systems
        - Enhances usability and security with additional features
        - Two key tools
            - SSH Agent
            - SFTP with chroot
    - *SSH Agent*
        - SSH agent stores decrypted private keys in memory
        - Reduces need to retype passphrase for every server connection
        - Command to load private key into agent
            - ssh-add ~/.ssh/id_rsa
                - ssh-add
                    - Loads the key
                - ~/.ssh/id_rsa
                    - The default private key location
        - SSH agent enables secure, efficient authentication across multiple systems
        - Ideal for enterprise environments with frequent system access
        - Reduces reliance on unprotected or insecure key practices
    - *SFTP with chroot*
        - SFTP allows encrypted file transfers via SSH

- Without restriction, users can navigate entire filesystem
- chroot restricts users to a specified directory
- Prevents access to files outside assigned areas
- Useful for managing contractors, clients, or junior users
- Configuration is made in /etc/ssh/sshd_config
- Example configuration

```
Match Group sftpusers
ChrootDirectory /home/sftp/%u
ForceCommand internal-sftp
X11Forwarding no
AllowTcpForwarding no
```

  - Match Group sftpusers
    - Targets users in the sftpusers group
  - ChrootDirectory /home/sftp/%u
    - limits users to their directories
  - ForceCommand internal-sftp
    - Ensures SFTP-only access
  - X11 and TCP forwarding
    - Disabled for security

- Summary
  - SSH Agent stores decrypted keys, reducing repeated authentication
  - Increases both security and convenience in multi-server environments
  - SFTP with chroot locks users to specific directories for secure file transfer
  - These tools support scalable, controlled, and secure remote access strategies

- **fail2ban**
  - Overview
    - *fail2ban*
      - Is a Linux security tool that bans IP addresses after repeated suspicious activity
    - Monitors log files for patterns such as failed login attempts
    - Applies timed bans to prevent unauthorized access
    - Key areas
      - Configuration
      - Monitoring logs
      - Triggering bans
      - Unblocking IPs
  - Configuration
    - Main config file
      - /etc/fail2ban/jail.conf
        - Do not edit directly
    - Custom config file
      - /etc/fail2ban/jail.local
    - Each section in jail.local is called a jail
    - Example SSH jail configuration

```
[sshd]
enabled = true
port = ssh
filter = sshd
logpath = /var/log/auth.log
maxretry = 5
bantime = 3600
```

- enabled = true
    - Activates the jail
- port = ssh
    - Specifies the SSH port
- filter = sshd
    - Applies the SSH filter
- logpath
    - Points to the monitored log file
- maxretry = 5
    - Bans IPs after 5 failed attempts
- bantime = 3600
    - Bans for 1 hour (3600 seconds)

■ Restart service to apply changes

- systemctl restart fail2ban

○ Monitoring Logs

■ Main log file for Fail2Ban

- /var/log/fail2ban.log
    - Logs show active jails, banned/unbanned IPs, timestamps

○ Triggering Bans

■ Simulate failed login attempts to test banning

■ SSH jail example

- Fail 5 logins to trigger a ban

■ Check jail status and banned IPs

- fail2ban-client status sshd
    - fail2ban-client
        - Is the CLI tool

        - status
            - Checks jail state
        - sshd
            - Is the jail name
    - Unblocking IPs
        - Remove banned IP using fail2ban-client
        - Syntax
            - fail2ban-client set [jail] unbanip [IP address]
        - Example
            - fail2ban-client set sshd unbanip 192.168.1.100
                - set
                    - Modifies the jail
                - unbanip
                    - Unblocks the specified IP
        - Check with fail2ban-client status sshd to confirm
    - Summary
        - fail2ban automatically bans IPs showing suspicious behavior
        - Configuration done in jail.local to protect specific services
        - Bans triggered after reaching failure threshold
        - Logs in /var/log/fail2ban.log show jail and IP activity
        - CLI tools provide jail status and manual control of bans

- **Avoid Unsecure Services**
    - Overview
        - Unsecure services transmit data in plain text across the network

- These include usernames, passwords, and files vulnerable to interception
- Key unsecure services
  - Telnet
  - FTP
  - TFTP
- Secure alternatives
  - SSH
  - SFTP
  - FTPS
  - SCP
- Disabling and removing unsecure services enhances system security

- *Telnet*
  - Provides remote system access through a command-line interface
  - Transmits all data, including credentials, in plain text
  - Vulnerable to interception by sniffing tools
  - Secure alternative: SSH (Secure Shell)
  - Disable Telnet
    - systemctl disable --now telnet.socket
  - Remove Telnet
    - apt remove telnet
    - yum remove telnet
- *FTP*
  - File Transfer Protocol used for uploading and downloading files
  - Sends usernames, passwords, and files without encryption
  - Risk of eavesdropping and credential theft
  - Secure alternatives

- SFTP (SSH File Transfer Protocol)
- FTPS (FTP Secure using SSL/TLS)

- Disable FTP (e.g., vsftpd)
  - systemctl disable --now vsftpd
- Remove FTP
  - apt remove vsftpd
  - yum remove vsftpd

- *TFTP*
  - Trivial File Transfer Protocol used for simple file transfers
  - Lacks authentication and encryption
  - Typically used in embedded systems or PXE boot environments
  - Serious security risk if accessible on public networks
  - Secure alternatives
    - SCP (Secure Copy Protocol)
    - SFTP
  - Disable TFTP
    - systemctl disable --now tftpd
  - Remove TFTP
    - apt remove tftp-hpa
    - yum remove tftp-server
- Summary
  - Telnet, FTP, and TFTP transmit data in plain text
  - Telnet should be replaced with SSH for encrypted remote access
  - FTP should be replaced with SFTP or FTPS for secure file transfers
  - TFTP should be restricted or replaced depending on use case

        - Always disable and remove unsecure services using systemctl and apt or yum

- **Disable Unused File Systems**
    - Overview
        - Linux supports a wide range of file systems by default
        - Many are outdated or unused, increasing the attack surface
        - Two key hardening steps
            - Block unused kernel modules
            - Review and disable entries in /etc/fstab
    - Unused Kernel Modules
        - Kernel modules extend Linux functionality, including file system support
        - Unused file system modules can be exploited
        - Common examples
            - cramfs
            - hfs
            - udf
        - Prevent module loading by adding lines in /etc/modprobe.d/
            - install cramfs /bin/false
                - Command tells system to execute /bin/false when attempting to load the module
                - Repeat for each unnecessary module
                - Prevents kernel from loading unused file systems even if requested
    - */etc/fstab*

- /etc/fstab controls which file systems are mounted at boot
- Includes local, external, and network file systems
- Outdated or unnecessary entries may create security risks
- Exposes volumes that should remain unmounted
- Use a text editor like nano or vi to review the file
- Disable lines by commenting them out
    - # /dev/sdb1 /mnt/backup ext4 defaults 0 2
        - Commented lines prevent mounting during boot
        - Helps ensure only trusted and active volumes are mounted

- Summary
    - Unused file system support increases system exposure
    - Block kernel modules for unused file systems with /etc/modprobe.d/
    - Review and comment out unnecessary /etc/fstab entries
    - Results in a cleaner, more secure file system configuration

- **Unnecessary SUID Permissions**
    - *SUID Bit*
        - SUID (Set User ID) bit is a special file permission applied to executable files
        - When set, a program runs with the privileges of the file's owner instead of the executing user
        - Commonly used by tools like passwd to allow regular users to update protected system files
        - Misuse or incorrect assignment of SUID can lead to security vulnerabilities

- Security Risk
  - If a script or program with SUID is insecure or unnecessary, attackers can exploit it to gain elevated access
  - SUID permissions on root-owned files can result in unauthorized privilege escalation
- Identification
  - Use ls -l to identify SUID files by locating an "s" in the owner's execute bit position
  - Example output
    - -rwsr-xr-x
- Removal
  - Remove the SUID bit with the command:
    - chmod u-s /path/to/file
  - Example
    - chmod u-s /usr/bin/somebinary
      - This ensures files run with the executing user's permissions instead of the file owner's

- *SUID Binaries*
  - SUID binaries are executables with the SUID bit set
  - Some are essential (e.g., /usr/bin/passwd), others may not be needed
  - Audit these binaries regularly to identify unnecessary or outdated tools
  - Discovery
    - Use the command below to list all SUID binaries
      - find / -perm -4000 -type f 2>/dev/null
        - Searches the entire file system for files with SUID set while suppressing errors

        - Mitigation
            - If a non-essential SUID binary is found (e.g., /usr/bin/rcp), either
                - Remove the SUID bit: chmod u-s /usr/bin/rcp
                - Uninstall the related package using the package manager
    - Summary
        - SUID permissions allow temporary privilege elevation for regular users
        - Incorrect usage of SUID increases the attack surface of the system
        - Regular auditing and removal of unnecessary SUID binaries is essential for system hardening
        - Managing SUID permissions properly helps prevent unauthorized access and maintains system security

- **Secure Boot**
    - *Secure Boot*
        - Secure Boot is a security feature that prevents unauthorized or malicious code from executing during system startup
        - Ensures only trusted software such as signed bootloaders and kernels can be loaded
        - Protects against low-level threats like bootkits and rootkits before the OS starts
        - Secure Boot depends on UEFI (Unified Extensible Firmware Interface) to verify digital signatures of boot components
    - *UEFI (Unified Extensible Firmware Interface)*
        - UEFI replaces the legacy BIOS and initializes hardware and boots the OS
        - Supports Secure Boot functionality

- Verifies boot components like the bootloader and OS kernel against a list of approved digital signatures
- If a component is unsigned or altered, the system refuses to boot
- Configuration via UEFI Firmware
    - Enter firmware setup during startup by pressing a key such as F2, Del, or Esc
    - Secure Boot can be enabled or disabled in the setup interface
    - Signature keys can be managed from this interface
    - Two key options are available
        - Use preloaded trusted keys
        - Enroll custom keys for unsigned or self-signed components

- Summary
    - Secure Boot ensures only digitally signed and trusted boot software can run during startup
    - It helps prevent low-level malware such as bootkits or rootkits from loading
    - Secure Boot operates through UEFI, which checks digital signatures before booting
    - UEFI settings allow enabling Secure Boot, managing trusted keys, or enrolling custom signatures
    - Secure Boot enhances Linux security from power-on, protecting the boot process from tampering

Automation and Orchestration

Objective 4.1: *Summarize the use cases and techniques of automation and orchestration in a Linux environment*

- **Ansible IaC Core Concepts**
  - Overview
    - Ansible automates system configuration and management through Infrastructure as Code (IaC)
    - Ansible is agentless and connects to systems using standard protocols
    - Four key core concepts
      - Agentless architecture
      - Inventory
      - Ad hoc mode
      - Modules
  - *Agentless Architecture*
    - No special software is required on managed systems
    - Connects using SSH for Linux and WinRM for Windows
    - Simplifies deployment and maintenance at scale
    - Avoids overhead of installing and updating background agents
    - Suitable for managing dozens or hundreds of systems efficiently
  - *Inventory*
    - Defines which systems Ansible will manage
    - Can be structured as
      - INI text files
      - YAML files

- Dynamic inventories from cloud platforms or CMDBs
- Example grouping
  - [web]
    - Group for web servers
  - [db]
    - Group for database servers
- Enables targeted actions on grouped hosts

- *Ad Hoc Mode*
  - Used for one-time, quick actions without writing a playbook
  - Syntax example
    - ansible all -m ping
      - Sends ping to all hosts in the inventory
  - Another example
    - ansible web -m service -a "name=nginx state=restarted"
      - Restarts nginx service on all hosts in the web group
        - Useful for validation, updates, and emergency fixes
        - Uses -m to specify the module
        - Uses -a to provide module arguments
- *Modules*
  - Each Ansible task is performed by a module
  - Examples of commonly used modules
    - apt
      - Manages packages on Debian-based systems
    - yum
      - Manages packages on Red Hat-based systems
    - user

        - Manages user accounts and properties
      - service
        - Starts, stops, or restarts services
      - copy
        - Copies files to managed systems
      - file
        - Creates, deletes, or modifies file and directory attributes
    - Modules simplify and standardize automation tasks
    - Enable scalable and repeatable configuration
  - Summary
    - Ansible enables automated system management using clear commands
    - Agentless design reduces complexity
    - Inventory files define what systems to control
    - Ad hoc mode runs quick tasks without writing playbooks
    - Modules execute defined tasks and actions efficiently across systems

- **Ansible IaC Advanced Usage**
  - Overview
    - Advanced Ansible features support automation in large-scale environments
    - Ad hoc commands are useful for quick tasks
    - Enterprise automation needs structured, repeatable, and modular tools
    - Three key components of advanced Ansible usage
      - Playbooks
      - Facts

        - Collections
    - *Playbooks*
        - Structured YAML files containing sets of Ansible tasks
        - Describe actions such as installing software, copying files, restarting services
        - Used to automate and enforce consistent configurations across systems
        - Ideal for managing hundreds of servers across dev, staging, and production
        - Eliminate manual workflows by creating repeatable and shareable instructions
        - Central element that transforms Ansible into a full automation tool
        - Code knowledge not required for the exam
    - *Facts*
        - System information collected automatically by Ansible
        - Include details like
            - IP address
            - Operating system
            - Memory
            - Disk space
        - Collected at the start of playbook execution
        - Used to create conditional tasks based on system properties
        - Enables dynamic and tailored configurations for each machine
        - Reduces need for manual input or hardcoded system values
    - *Collections*
        - Packages that group related automation components
        - Include

        - Roles
        - Modules
        - Plugins
    - Help reuse and share tools for consistent automation practices
    - Often downloaded for managing cloud providers like AWS or Azure
    - Support modularity and standardized team practices
    - Allow scaling and organizing Ansible automation projects
  - Summary
    - Advanced Ansible features improve efficiency and scalability in enterprise environments
    - Playbooks group tasks into organized scripts for consistent deployment
    - Facts enable data-driven decision-making during automation
    - Collections promote reuse, modularity, and shared best practices
    - These features together empower administrators to manage complex systems reliably and at scale

- **Puppet Core Usage**
  - Overview
    - Puppet automates system configuration using a model-driven approach
    - Configurations are described and enforced automatically by agents
    - Three core building blocks of Puppet configuration management
      - Facts
      - Classes
      - Modules

- *Facts*
  - Puppet is agent-based by default
  - Each managed node runs a Puppet agent
  - Agents collect system information known as facts
  - Facts include
    - Operating system
    - Hostname
    - IP address
    - Memory
  - Facts are gathered using the Facter tool
  - Facts guide configuration decisions during check-ins with the Puppet server
  - Differences from Ansible
    - Puppet
      - Facts collected automatically and continuously
    - Ansible
      - Facts collected only on task execution over SSH
  - Puppet model is state-driven and continuous
  - Ansible model is task-driven and on-demand
- *Classes*
  - Classes group related configuration tasks into logical units
  - Example
    - A class might install and configure a web server
  - Promote reusability and maintainability across systems
  - Equivalent to playbooks in Ansible
  - Puppet uses a declarative style

- Describes desired system state
- Puppet determines how to achieve that state

- Classes enable centralized and consistent configuration across many machines

- *Modules*
  - Packages containing Puppet code for a specific task or service
  - Modules include
    - One or more classes
    - Templates
    - Files
    - Custom facts
  - Organize and reuse configuration code in larger environments
  - Puppet Forge
    - Official site for downloading community and vendor modules
  - Similar to Ansible collections
  - Modules are the primary method for structuring and sharing Puppet workflows
- Summary
  - Puppet automates system setup using a model-driven architecture
  - Facts collected by Puppet agents help tailor configurations based on system state
  - Classes organize sets of configuration rules into reusable, logical units
  - Modules bundle classes, templates, and resources to support code reuse and consistency
  - Together, these components enable scalable, reliable, and consistent configuration management across many systems

- **Puppet Advanced Usage**
    - Overview
    - Puppet operates using a secure, scalable infrastructure ideal for large enterprise environments
    - Two foundational components
        - Agent Architecture for automated system management
        - Certificate-Based Authentication for secure communication
    - *Agent and Agentless Architecture*
        - Puppet is agent-based by default
        - Each managed machine runs the Puppet agent
        - The Puppet agent
            - Communicates with the central Puppet server (Puppet master)
            - Periodically retrieves configuration instructions
            - Applies configurations automatically in the background
        - Advantages of agent-based model
            - Continuous enforcement of system state
            - Scalable for managing large numbers of machines
            - Reduces need for manual intervention
        - Ansible comparison
            - Agentless
            - Uses SSH or WinRM
            - Pushes changes immediately during task execution
            - Ideal for ad hoc or one-time changes

- *Certificates*
  - Puppet uses digital certificates to secure agent-server communication
  - Initial agent communication
    - Sends a certificate signing request to the Puppet server
    - Server must approve and sign the certificate
  - Purpose
    - Ensures only trusted machines can access configuration data
    - Prevents unauthorized systems from joining the Puppet network
  - Admin responsibilities
    - Approve new certificate requests
    - Revoke or clean up outdated or unauthorized certificates
- Summary
  - Puppet's agent-based architecture supports automated, scheduled configuration management
  - Certificates ensure trusted and encrypted communication between agents and the Puppet server
  - Together, these mechanisms make Puppet a reliable and secure tool for large-scale configuration management

- **OpenTofu Core Usage**
  - *OpenTofu*
    - OpenTofu is an open-source Infrastructure as Code (IaC) tool
    - Used to manage and automate cloud infrastructure with code
    - Replaces manual dashboard interactions with consistent, version-controlled configurations

- Supports platforms like AWS, Azure, Google Cloud, and on-premise infrastructure
- Applies infrastructure changes via APIs during the apply step

- Core Concepts
  - *Provider*
    - A provider connects OpenTofu to a cloud platform or service
    - Acts as a plugin that understands the platform's APIs
    - Used to create, modify, or delete infrastructure such as virtual machines, networks, and databases
    - Multiple providers can be used across environments and accounts
    - Differs from
      - Ansible, which uses SSH to push configurations
      - Puppet, which uses agent/server pull model
    - Essential first step in any OpenTofu configuration
  - *Resource*
    - A resource is a specific component of infrastructure (e.g., VM, firewall rule, storage bucket)
    - Defined in configuration files and linked to a provider
    - Enables version-controlled, consistent deployments and rollbacks
    - Focuses on provisioning cloud services, unlike
      - Puppet classes, which manage state at OS level
      - Ansible modules, which manage system tasks
  - *State*
    - State keeps track of the current status of all managed infrastructure
    - Helps OpenTofu determine what to add, update, or delete

- Functions as the source of truth for infrastructure
- Typically stored remotely for team collaboration and change control
- Distinct from
    - Ansible (task-driven and stateless)
    - Puppet (model-driven with enforced state but no separate state record)

- Summary
    - OpenTofu enables infrastructure automation through code
    - Core elements include
        - Provider
            - To interface with cloud platforms
        - Resource
            - To define infrastructure components
        - State
            - To track current infrastructure and changes
    - Enables consistent, collaborative infrastructure management across environments

- **OpenTofu Advanced Usage**
    - OpenTofu API Integration
        - OpenTofu interacts with cloud platforms through their APIs
        - APIs (Application Programming Interfaces) are used to create, update, or delete infrastructure

        - Providers send structured API requests rather than executing commands on individual machines
        - OpenTofu's model is top-down and declarative
    - API Communication
        - Configuration files define desired infrastructure state
        - Running tofu apply triggers
            - Comparison of current state file to desired configuration
            - API calls to cloud platforms to reconcile any differences
        - Examples of API operations
            - Create a virtual machine
            - Open a network port
            - Delete a storage bucket
    - Comparison to Other Tools
        - OpenTofu vs Ansible
            - OpenTofu uses APIs to manage cloud infrastructure
            - Ansible uses SSH or WinRM to push configuration to machines
        - OpenTofu vs Puppet
            - Puppet uses agents to manage internal system state
            - OpenTofu uses APIs to control infrastructure directly at the platform level
    - Enterprise Benefits
        - API model supports
            - Version control
            - Automation pipelines
            - Collaborative workflows
        - Improves visibility with audit logs from cloud platforms

        - Reduces manual access to individual systems
        - Enables secure, consistent, and real-time changes to infrastructure
    - Summary
        - OpenTofu uses providers to send API requests to cloud services like AWS, Azure, and Google Cloud
        - These API interactions create and manage infrastructure without manual system access
        - The tofu apply process compares state and executes only necessary changes
        - API-driven architecture supports automation, scalability, and auditability in enterprise environments

- **Unattended Deployment**
    - Overview of Unattended Deployment
        - Automates system installation and configuration without manual input
        - Reduces time and human error in large-scale deployments
        - Common tools include
            - Kickstart
            - Cloud-init
    - *Kickstart*
        - Used for automating installation of Red Hat-based systems such as RHEL and Rocky Linux
        - A Kickstart file contains pre-defined answers and settings for the installer
        - Parameters include language, disk setup, network configuration, and packages

- Installation is launched using boot prompt syntax
  - linux ks=<location_of_kickstart_file> inst.repo=<installation_source>
- Example command
  - linux ks=http://192.168.1.10/kickstart/ks.cfg inst.repo=http://192.168.1.10/rhel8
    - ks=
      - Points to the Kickstart file
    - inst.repo=
      - Specifies the installation file source
- System installs automatically using the Kickstart instructions

- *Cloud-init*
  - Standard for automating first-boot configuration of cloud-based Linux instances
  - Executes tasks like hostname setting, SSH key setup, package installation, and script execution
  - Reads a YAML configuration file passed in as user data
  - User data is passed when launching an instance via CLI or cloud dashboard
  - Example AWS CLI command
    - aws ec2 run-instances --image-id ami-0abcdef1234567890 --instance-type t2.micro --user-data file://init-script.yaml
  - --user-data flag provides the Cloud-init file to the instance
- Summary
  - Unattended deployment enables automatic, consistent, and hands-free system setup

        - Kickstart is ideal for Red Hat-based traditional environments
        - Cloud-init is designed for cloud platforms and executes on first boot
        - Both tools reduce setup time and improve deployment efficiency at scale

- **Foundational CI/CD Concepts**
    - Overview of CI/CD
        - CI/CD stands for Continuous Integration and Continuous Deployment
        - Brings structure and automation to software development
        - Helps prevent issues like lost changes or conflicting updates in team-based projects
        - Foundational concepts include version control and pipelines
    - *Version Control*
        - Tracks changes to files over time for collaboration and rollback
        - Git is the most widely used version control system
        - Allows local commits and remote updates to shared repositories
        - Example commands
            - git commit -am "Update configuration file"

              git push origin main
                - git commit -am
                    - Creates a local snapshot with a message
                - git push origin main
                    - Uploads changes to the main branch of the remote repository
        - Enables parallel development with complete history tracking
    - *Pipelines*

- Automate the process from commit to deployment
- Common stages include testing, security scans, builds, and deployment
- Triggered automatically after code is pushed to version control
- Example GitLab CLI command
    - glab pipeline run --ref main
        - glab
            - Is the GitLab CLI
        - pipeline run
            - Starts the CI pipeline
        - --ref main
            - Specifies the main branch as the pipeline source
- Reduces manual effort and provides fast feedback
- Enhances consistency in enterprise environments

- Summary
    - CI/CD streamlines modern software development by integrating automation and collaboration tools
    - Version control enables efficient teamwork and change tracking
    - Pipelines ensure changes are tested and deployed reliably
    - Together, they form the foundation of scalable and dependable development workflows

- **Modern CI/CD Approaches**
    - Overview of Modern CI/CD
        - Modern CI/CD approaches improve software delivery speed, security, and reliability

- Integrate automation and best practices throughout the development lifecycle
- Key concepts include Shift Left Testing, DevSecOps, and GitOps

- *Shift Left Testing*
  - Moves testing earlier in the software development cycle
  - Replaces traditional late-stage testing with early, automated test execution
  - Common tests include
    - Unit tests
    - Security scans
    - Configuration validations
  - Integrates with version control systems to trigger tests on every code push
  - Tools such as Jenkins and GitLab CI embed test stages into CI/CD pipelines
  - Reduces costs and downtime by catching issues early
- *DevSecOps*
  - Combines Development, Security, and Operations into a unified approach
  - Embeds security checks directly into CI/CD pipelines
  - Automates scanning for vulnerabilities, compliance, and misconfigurations
  - Tools such as Trivy and Snyk scan containers and code dependencies
  - Promotes shared responsibility among developers, security teams, and administrators
  - Speeds up delivery by integrating security without creating bottlenecks
- GitOps
  - Manages infrastructure and deployments using Git as the source of truth

        - Updates to infrastructure are made via Git repository changes
        - Automated tools apply updates to environments when changes are committed
        - Tools such as Argo CD and Flux sync environments with Git repositories
        - Reduces configuration drift and simplifies rollback by reverting commits
        - Enhances visibility, traceability, and consistency in enterprise environments
    - Summary
        - Modern CI/CD practices focus on speed, automation, security, and reliability
        - Shift Left Testing catches bugs and issues early in development
        - DevSecOps integrates security scanning and compliance into every CI/CD stage
        - GitOps manages infrastructure declaratively using Git for visibility and control
        - These approaches collectively support scalable, secure, and efficient software delivery

- **Kubernetes Core Workloads for Deployment Orchestration**
    - Overview of Kubernetes Core Workloads
        - *Kubernetes*
            - Is an open-source platform for automating deployment, scaling, and management of containerized applications
        - Core workloads include
            - Pods

- Deployments
- Services

- These components orchestrate application availability, scalability, and accessibility

- *Pods*
  - Smallest and most basic unit in Kubernetes
  - Consist of one or more containers that run on the same host
  - Containers in a Pod share network namespace and storage volumes
  - Ideal for tightly coupled containers such as an application and its helper process
  - Serve as the fundamental building block for deploying applications in Kubernetes
  - Example usage includes running a containerized NGINX web server with defined port and resource limits
- *Deployments*
  - Manage the lifecycle and scaling of Pods
  - Act as controllers to ensure the desired number of Pods are running and up to date
  - Provide support for rolling updates to minimize downtime
  - Automatically replace failed Pods and scale applications as needed
  - Example usage includes updating a version of an internal app while maintaining availability
- *Services*
  - Provide stable endpoints for accessing Pods
  - Abstract the dynamic nature of Pod IPs which can change upon restarts
  - Ensure reliable communication between Pods or with external clients

        - Enable load balancing and service discovery within the Kubernetes cluster
        - Example usage includes directing incoming traffic from a company portal to the appropriate backend Pods
    - Summary
        - Kubernetes automates container orchestration across clusters
        - Pods host application containers and represent the deployment unit
        - Deployments manage scaling and updating of Pods with high reliability
        - Services offer persistent access points to ensure stable communication and connectivity
        - Together, Pods, Deployments, and Services support scalable and fault-tolerant enterprise applications

- **Kubernetes Configuration**
    - Overview of Kubernetes Configuration
        - Configuration includes environment settings, database details, and API keys required by applications
        - Kubernetes provides mechanisms to manage configuration securely and separately from application code
        - Core configuration tools include Variables, ConfigMaps, Secrets, and Volumes
    - *Variables*
        - Environment variables are used to pass simple settings into containers
        - Defined directly in Pod or Deployment specifications
        - Used for values such as environment names or feature flags
        - Example usage includes setting ENVIRONMENT=production

- Suitable for small, static values
- Often used in combination with other configuration tools in enterprise environments

- *ConfigMaps*
  - Store non-sensitive configuration data such as full config files or multiple environment variables
  - Managed as separate Kubernetes objects
  - Help decouple configuration from application definitions
  - Improve reusability and version control
  - Example usage includes shared logging settings or service URLs across multiple applications
  - Updating the ConfigMap updates all linked workloads
- *Secrets*
  - Designed for storing sensitive information such as passwords, tokens, or certificates
  - Encoded using base64 by default
  - Encryption at rest can be enabled via EncryptionConfiguration
  - Delivered to containers as environment variables or mounted files
  - Access to Secrets is controlled using Role-Based Access Control (RBAC)
  - RBAC defines which users or services can read or modify Secrets and other resources
- *Volumes*
  - Provide persistent storage beyond the container lifecycle
  - Essential for applications that maintain state like databases or content systems

        - Support various backends such as hostPath, Amazon EBS, Google Persistent Disk, and Azure Disks
        - PersistentVolumeClaim (PVC) allows requesting storage with specific size and access mode
        - Volumes remain intact during restarts, reboots, or pod migrations
        - Used for storing logs, uploaded files, or application data
    - Summary
        - Kubernetes configuration tools allow flexible and secure application behavior across environments
        - Environment variables offer basic, inline configuration
        - ConfigMaps enable organized, reusable, non-sensitive settings
        - Secrets protect sensitive data with encoding, encryption, and access controls
        - Volumes ensure data persistence for stateful applications
        - Together, these tools empower administrators to manage complex, scalable applications reliably within a Kubernetes cluster

- **Docker Swarm Core Workloads for Deployment Orchestration**
    - Overview of Docker Swarm
        - Docker Swarm orchestrates containerized applications across multiple machines
        - Simplifies deployment, scaling, and maintenance of containers
        - Three core components include Nodes, Tasks, and Services
    - *Nodes*

- A node is any machine (physical or virtual) running the Docker Engine within a Swarm
- Nodes are categorized as manager nodes or worker nodes
- Manager nodes handle orchestration and assign tasks
- Worker nodes execute the assigned tasks
- Administrators manage node health, connectivity, and resource availability
- Comparable to Kubernetes nodes but simpler in configuration

- *Tasks*
  - A task is a single running container instance assigned by Swarm
  - Created automatically when a service is deployed
  - Each task maps to exactly one container
  - Swarm monitors and replaces failed tasks automatically
  - No need for manual restarts or scaling interventions by administrators
  - Functionally similar to Kubernetes pods but limited to one container per task
- *Services*
  - A service defines how an application should run within Docker Swarm
  - Specifies container image, number of replicas, exposed ports, and environment variables
  - Administrators can scale services by adjusting replica counts
  - Built-in support for load balancing and rolling updates
  - Offers core deployment capabilities with lower complexity than Kubernetes Deployments
- Summary

- Docker Swarm coordinates and manages containers efficiently across a cluster
- Nodes form the infrastructure layer, categorized into manager and worker roles
- Tasks are individual containers managed and monitored by the system
- Services define the deployment parameters and maintain high availability
- Provides an accessible solution for enterprise teams needing orchestration without extensive complexity

- **Docker Swarm Configuration**
  - Overview
    - Docker Swarm configuration enables control over communication and scaling of services across multiple nodes
    - Key configuration features include overlay networks and scaling replicas
    - Ensures secure, efficient, and highly available container deployments in enterprise environments
  - *Networks*
    - Containers in a Swarm need to communicate across nodes
    - Overlay networks enable inter-container communication across hosts
    - Defined using the overlay driver in docker-compose.yml
    - Example

```
networks:
  frontend:
    driver: overlay
```

- Enables
  - Secure cross-node communication
  - Isolation of services
  - Enforcement of internal network policies
- Automatically created and managed by Docker during stack deployment

- *Scale*
  - Scale determines how many container replicas of a service are run
  - Defined under the deploy section in docker-compose.yml
  - Example

```
services:
  web:
    image: nginx
    deploy:
      replicas: 3
```

  - Docker Swarm
    - Distributes replicas across available nodes
    - Maintains high availability by replacing failed replicas
  - CLI alternative
    - docker service scale web=5
- Summary
  - Docker Swarm configuration ensures distributed applications are scalable and network-aware
  - Networks use overlay drivers to enable container communication across nodes
  - Scaling defines the number of service replicas to maintain performance and availability

- Together, these tools support secure and resilient production deployments

- **Docker/Podman Compose for Deployment Orchestration**
  - Overview
    - Simplifies deployment of multi-container applications using a single configuration file
    - Useful for applications with multiple services such as web servers, databases, and load balancers
    - Reduces complexity and supports automation in development and production environments
    - Key elements
      - Compose files
      - Up and down commands
      - Logs
  - *Compose Files*
    - YAML-formatted configuration files usually named docker-compose.yml
    - Defines containers, images, ports, networks, and volumes for the application
    - Ensures consistency across development, staging, and production
    - Example docker-compose.yml

```yaml
version: '3.8'
services:
  web:
    image: nginx
    ports:
      - "80:80"
  app:
    image: my-app:latest
    environment:
      - ENV=production
    depends_on:
      - db
  db:
    image: postgres
    volumes:
      - db-data:/var/lib/postgresql/data
```

- Services
  - web
    - Uses NGINX and listens on port 80
  - app
    - Custom application with ENV variable
  - db
    - PostgreSQL database with volume persistence
- depends_on
  - Ensures database starts before the app
- volumes
  - Defines persistent storage with db-data

- Tools
  - Docker Compose
    - Works with Docker engine, widely supported
  - Podman Compose
    - Works with Podman, rootless and security-focused
  - Both tools support the same Compose file format

- *Up and Down Commands*
  - Start all containers with
    - docker compose up

- podman-compose up
- Commands read the Compose file, build images if needed, and start all services
- Stop and remove containers with
  - docker compose down
  - podman-compose down
- Removes default network, but not named volumes unless --volumes is added
- Provides simple full-stack deployment and cleanup with minimal commands

- *Logs*
  - View container output using
    - docker compose logs
    - podman-compose logs
  - View specific service logs
    - docker compose logs web
  - Follow logs in real-time
    - docker compose logs --follow web
  - Includes standard output and error from containers
  - Useful for
    - Error detection
    - Startup monitoring
    - Deployment verification
  - Docker integrates easily with centralized logging solutions
  - Podman may require additional setup for centralized logging
- Summary

- Compose tools enable declarative, consistent orchestration using docker-compose.yml
- up and down commands manage the full lifecycle of multi-container apps
- Logs allow centralized viewing of container output for diagnostics and validation
- Compose works with both Docker (feature-rich) and Podman (security-enhanced) environments

Automated Tasks with Shell Scripting

Objective 4.2: *Given a scenario, perform automated tasks using shell scripting*

- **Parameter Expansion**
  - Overview of Parameter Expansion
    - Parameter expansion allows insertion of variable values into commands or scripts
    - Uses syntax ${var} to dynamically reference a variable's value
    - Enables scripts and commands to be more flexible, reusable, and easier to maintain
  - Basic Syntax
    - ${var} is used in shell environments like Bash
    - Substitutes the value of var into a command or script
    - Commonly used in shell scripts and one-liner terminal commands
    - Eliminates hard-coded values and improves readability
  - Example: Basic Directory Navigation
    - Variable assignment
      - dir_name="/var/log"
    - Using the variable with parameter expansion
      - cd ${dir_name}
    - Expands to
      - cd /var/log
  - Example: Dynamic Backup Filename with Date
    - Variable assignment using date command

        - today=$(date +%F)
    - Using the variable to build a file name
        - cp /etc/hosts /backup/hosts-${today}.bak
    - Expands to
        - cp /etc/hosts /backup/hosts-2025-05-28.bak
- Summary
    - Parameter expansion uses ${var} to insert variable values into scripts and commands
    - Enhances automation by allowing dynamic file paths, names, or command arguments
    - Helps administrators create organized, consistent, and date-based filenames
    - Supports flexible directory navigation and reusable code structures

- **Command Substitution**
    - Overview
        - Allows shell to run a command and insert its output into another command or script
        - Enables dynamic automation and scripting
        - Two core components
            - Literal strings using single quotes
            - Command substitution using $(...)
    - *Single-Quoted Strings: 'foo'*
        - Everything inside is treated as literal text
        - No variable expansion, no command substitution

- Shell prints text exactly as written
- Useful for logs, templates, or messages without shell interpretation
- Example
  - Command
    - echo 'Warning: $PATH is not set'
  - Output
    - Warning: $PATH is not set

- *Command Substitution: $(foo)*
  - Tells shell to run the command inside parentheses
  - Replaces $(...) with the command's output
  - Useful for dynamic values in scripts or commands
  - Example
    - Command
      - mkdir /backup/$(date +%F)
    - If today is 2025-05-28, it creates /backup/2025-05-28
- Summary
  - 'foo' prevents evaluation
    - Used for literal output
  - $(foo) triggers command substitution
    - Outputs result of command
  - Important distinction for writing effective shell scripts
  - Always check your quote usage depending on your intent

- **Subshell Execution**
  - Overview

- A subshell is a separate child process created by the shell to run commands in isolation
- Commands inside a subshell do not affect the environment of the parent shell
- Syntax for subshell execution is wrapping commands in parentheses
- Example
  - (command1; command2; ...)

- Isolating Environment Changes
  - Subshells can be used to execute temporary changes without affecting the main session
  - Directory changes, variable assignments, and other environment modifications inside the subshell remain local
  - Example
    - (pwd; cd /etc; ls; pwd)
- Use with Command Substitution
  - Subshells can be combined with command substitution to capture output while isolating changes
  - Output from a subshell can be stored in a variable and used later in the script
  - Example
    - result=$((echo "Task started"; sleep 1; echo "Task complete"))
- Using the Captured Output
  - The captured result can be redirected to logs or evaluated in scripts
  - Helps with scripting logic, logging, and conditional processing
  - Example
    - echo "$result" >> /var/log/maintenance.log

- echo "Maintenance Summary: $result"
- if [[“$result" == *"Task complete"*]]; then echo "All operations finished successfully."; fi

- Summary
  - A subshell in Linux is created using parentheses to isolate command execution
  - It prevents temporary changes from affecting the main shell session
  - Useful in scripting for testing, isolation, and capturing outputs dynamically
  - Subshells maintain a clean environment and enable reliable scripting practices

- **Functions**
  - Reusability
    - Functions package a set of commands under a single name for repeated use
    - Reduces redundancy and makes scripts easier to maintain
    - Generic syntax in Bash
    - Example

```
function_name() {
  commands go here;
}
```

    - Administrator example for checking disk usage in /var with a timestamp

```
check_disk() {
  echo "$(date +%F) - $(du -sh /var)";
}
```

- Return Values
  - Bash functions cannot return strings or structured data directly
  - Return of text data in Bash is done with echo, but this can be clunky
  - Python functions allow returning strings, numbers, lists, dictionaries using return
  - Python function syntax
  - Example

```
def function_name():
  # logic here
  return value
```

  - Example checking syslog file size

```
import os

def check_log_size():
  size_mb = os.path.getsize("/var/log/syslog") / (1024 * 1024)
  if size_mb > 100:
    return "Alert: syslog exceeds 100MB"
  return "Log size is normal"
```

- Variable Scope
  - By default, Bash variables are global
  - Use local keyword to restrict variable scope to inside the function
  - Example

```
build_path() {
  local dir="/etc"
  local file="hosts"
  echo "$dir/$file"
}
```

  - dir and file are not accessible outside the function

        - Local scope prevents interference with global script variables
    - Summary
        - Functions make scripts more reusable, organized, and easier to maintain
        - Reusability allows defining a task once and calling it multiple times
        - Return values allow functions to send back data for further use; Python offers flexible return types
        - Variable scope controls where a variable is valid; using local in Bash restricts it to the function
        - Together, these practices lead to efficient, safe, and clean scripting

- **Internal Field Seperator / Output Field Separator**
    - Avoiding Word Splitting
        - Shell splits variable content into words using spaces, tabs, and newlines by default
        - Quoting variables or using printf avoids unintended splitting
        - Safer method to output a variable with internal whitespace
        - Example
            - printf '%s\n' "$variable"
        - Example storing a file path with spaces
            - file_path="My Documents/Project Report.txt"
        - Naive usage
            - cat $file_path fails due to splitting
        - Correct usage
            - printf '%s\n' "$file_path"
        - Output

        - My Documents/Project Report.txt
    - Controlling Input Splitting
        - Input splitting can be controlled using IFS (Internal Field Separator)
        - IFS defines what character triggers field boundaries during read
        - Generic pattern
            - IFS=<delimiter> read -r var1 var2 ... <<< "$text"
        - Example with comma-separated values
            - IFS=',' read -r name age city <<< "alice,24,Denver"
        - Variables
            - name="alice"
            - age="24"
            - city="Denver"
        - Output
            - echo "$name is $age years old and lives in $city"
        - Result
            - alice is 24 years old and lives in Denver
    - Output Formatting
        - *Output Field Separator (OFS)*
            - Is used in tools like awk for formatted output
        - Sets the character to place between output fields
        - Common awk pattern
            - awk 'BEGIN{OFS="<delimiter>"} {print $1,$2,...}' file
        - Example converting /etc/passwd to CSV
        - awk 'BEGIN{OFS=","} {print $1,$3,$4}' /etc/passwd | head -n 3
        - Explanation
            - OFS is set to comma

- $1, $3, $4 refer to username, UID, GID

- Output

```
root,0,0
daemon,1,1
bin,2,2
```

- Summary
  - IFS controls where shell breaks input text into fields
  - OFS defines how fields are joined during output, especially in tools like awk
  - To avoid unintended breaks, quote variables or use printf
  - For intentional splitting, set IFS or use delimiters in tools like awk
  - Format output using OFS or printf to ensure structured, readable data

- **Conditional Statements**
  - If Conditional Statement
    - if evaluates a true-or-false condition to decide which code branch to run
    - Syntax structure

```
if [ condition ]; then
    commands
else
    alternate commands
fi
```

    - Example backup logic using if

```
if [ -f "$SOURCE" ]; then
    cp "$SOURCE" /backup
    echo "Backup completed"
else
    echo "File not found"
fi
```

- [-f "$SOURCE"] checks if $SOURCE is a regular file
- cp copies the file and echo prints status
- fi ends the conditional block
- Used for tasks like service checks, disk space verification, or variable validation

- Case Conditional Statement
  - case compares a single variable against multiple patterns for multi-branch logic
  - Syntax structure

```
case "$variable" in
    pattern1)
        commands
        ;;
    pattern2)
        commands
        ;;
    *)
        default commands
        ;;
esac
```

  - Example service control logic using case

```
action="$1"
case "$action" in
    start)
        systemctl start myapp
        ;;
    stop)
        systemctl stop myapp
        ;;
    status)
        systemctl status myapp
        ;;
    *)
        echo "Usage: $0 {start|stop|status}"
        ;;
esac
```

- $1 is the first positional argument passed to the script

- $action stores the value of $1 for evaluation
- Pattern match triggers a specific command block followed by ;;
- *) serves as the default catch-all case
- Used for clean handling of command-line options in scripts

- Summary
  - if is used for binary true-or-false decisions and supports then, else, and fi structure
  - case evaluates one variable against multiple possible values using esac block
  - if is ideal for single-condition checks such as file existence
  - case is preferred for handling multiple command options or script modes
  - Both are essential for building responsive, logic-driven scripts

- **Looping Statements**
  - *For Loop*
    - Used to repeat a task for a specific number of times or over a list of known items
    - Syntax structure

```
for variable in list
do
    commands
done
```

    - Example loop over a name list

```
for name in Alice Bob Charlie
do
    echo "Hello, $name!"
done
```

- Loops over each value (Alice, Bob, Charlie), assigns it to name, and prints a greeting
- Ideal when the total number of iterations is known at the start

- *While Loop*
  - Used to repeat a task as long as a specified condition remains true
  - Syntax structure

```
while [ condition ]
do
    commands
done
```

  - Example counter loop

```
counter=1
while [ $counter -le 3 ]
do
    echo "Count is $counter"
    ((counter++))
done
```

    - Starts with counter at 1, prints and increments until counter exceeds 3
    - Best suited for tasks with unknown total repetitions but a clear exit condition

- *Until Loop*
  - Similar to a while loop but runs until a condition becomes true (opposite logic)
  - Syntax structure

```
until [ condition ]
do
    commands
done
```

        - Example using the same counter logic

```
counter=1
until [ $counter -gt 3 ]
do
    echo "Count is $counter"
    ((counter++))
done
```

            - Repeats until counter becomes greater than 3, output identical to while example
            - Choice between while and until depends on how the condition is framed
    - Summary
        - For loop is used for known quantities or items
        - While loop is used when looping continues while a condition is true
        - Until loop is used when looping continues until a condition is true
        - All loops reduce code repetition and enhance script automation and logic

- **Interpreter Directive**
    - Overview of Interpreter Directive
        - The interpreter directive tells the system which program should execute a script
        - It appears as the first line in the script and begins with #!, known as a "shebang"
        - It specifies the path to the interpreter such as /bin/bash for Bash scripts
    - Purpose and Importance
        - Ensures the correct interpreter is used to process the script's commands

- Critical for proper execution when running the script directly from the shell
- Without it, the system may not know how to interpret the script's syntax

- Example: Bash Script with Shebang
  - Script filename: hello.sh
  - Script content
    - #!/bin/bash
    - echo "Hello, world!"
      - The line #!/bin/bash tells the system to use the Bash shell
      - The command echo "Hello, world!" is interpreted by Bash and executed
- Summary
  - The interpreter directive starts with #! followed by the full path of the interpreter
  - Common interpreter paths include
    - #!/bin/bash for Bash scripts
    - #!/usr/bin/python3 for Python scripts
    - #!/bin/sh for POSIX-compliant shell scripts
  - The shebang is essential for correct script execution and interpreter selection

- **Numerical Comparisons**
  - -eq and -ne
    - *-eq*
      - Checks if two numbers are equal

- *-ne*
  - Checks if two numbers are not equal
- Syntax is placed inside square brackets
- Example

```
if [ $number -eq 5 ]; then
    echo "The number is exactly 5"
fi
if [ $number -ne 10 ]; then
    echo "The number is not 10"
fi
```

  - -eq returns true when numbers match
  - -ne returns true when numbers differ
  - Useful for simple numeric comparisons and validations

- -gt and -lt
  - *-gt*
    - Means greater than
  - *-lt*
    - Means less than
  - Syntax is placed inside square brackets
  - Example

```
if [ $value -gt 5 ]; then
    echo "The value is greater than 5"
fi
if [ $value -lt 10 ]; then
    echo "The value is less than 10"
fi
```

    - Helpful for comparing values against thresholds
    - Allows condition-based scripting for high/low limits
- -ge and -le

- *-ge*
  - Means greater than or equal to
- *-le*
  - Means less than or equal to
- Syntax is placed inside square brackets
- Example

```
if [ $score -ge 70 ]; then
    echo "Score is passing (70 or higher)"
fi
if [ $score -le 100 ]; then
    echo "Score is within the maximum range"
fi
```

  - Includes edge values in comparisons
  - Useful for upper and lower bounds in logic

- Summary
  - -eq equal to
  - -ne not equal to
  - -gt greater than
  - -lt less than
  - -ge greater than or equal to
  - -le less than or equal to
  - All numeric comparisons must be placed inside square brackets in Bash
  - Used in conditional blocks to control script behavior based on numeric values

- **Redirection String Operators**
  - *> Redirection Operator*
    - Used to send the output of a command to a file
    - Creates the file if it doesn't exist
    - Overwrites the file if it already exists
    - Commonly used for logging or storing output
    - Must be used carefully to avoid unintended data loss
    - Example

```
number=5
if [ $number -gt 3 ]; then
    echo "The number is greater than 3" > result.txt
fi
```

      - Stores the message in result.txt if condition is true
  - *< Redirection Operator*
    - Used to take input from a file and provide it to a command
    - Useful for automated scripts requiring predefined input
    - Feeds file contents into a script without manual typing
    - Example

```
read number < input.txt
if [ $number -lt 10 ]; then
    echo "The number is less than 10"
fi
```

      - Reads the value from input.txt and stores it in number
      - Compares the value to 10 and prints a message if condition is met
  - Summary
    - > sends output to a file
    - < reads input from a file
    - > is used for saving command results

- < is used for automating input without manual interaction
- Together, these operators improve script automation and file handling

- **Comparison String Operators**
  - ==, =, and =~
    - ==
      - Used in [[ ]] to check if two strings are equal
    - =
      - Used in [ ] to perform the same equality check
    - =~
      - Used in [[ ]] to match a string against a regular expression
    - [[$word == "hello"]] evaluates true if the word equals "hello"
    - [[$word =~ ^h]] evaluates true if the word starts with "h"
    - Example

```
word="hello"
if [[ $word == "hello" ]]; then
    echo "The word is exactly hello"
fi
if [[ $word =~ ^h ]]; then
    echo "The word starts with h"
fi
```

  - *!=*
    - Used to check if two strings are not equal
    - Evaluates true when the strings do not match
    - Helpful for verifying input or detecting unexpected values
    - Example

```
status="offline"
if [ $status != "online" ]; then
    echo "The system is not online"
fi
```

- *<= and >=*
  - Perform alphabetical (lexicographical) comparisons
  - <=
    - Checks if the string is less than or equal to another
  - >=
    - Checks if the string is greater than or equal to another
  - Must be used inside [[ ]]
  - Example

```
item="banana"
if [[ $item >= "apple" ]]; then
    echo "$item comes after or is equal to apple"
fi
if [[ $item <= "cherry" ]]; then
    echo "$item comes before or is equal to cherry"
fi
```

- Summary
  - == and = compare strings for equality
  - != checks for inequality
  - =~ allows pattern matching using regular expressions
  - <= and >= compare strings based on alphabetical order
  - Use [[ ]] for extended pattern support and lexicographical comparisons
  - Proper string comparison allows scripts to make accurate, text-based decisions

- **Regular Expressions**
  - Overview
    - Used to check if a string matches a specific pattern
    - Allows input validation, filtering, and data structure checks in scripts
    - Format in Bash
      - [[$variable =~ pattern]]
    - Uses =~ operator inside double square brackets
    - More powerful than exact string comparison
  - Command Format: [[$foo =~ regex]]
    - [[...]]
      - Enables Bash's extended test capabilities
    - $foo
      - Holds the string to be tested
    - =~
      - Tells Bash to interpret the right-hand side as a regular expression
    - Useful for checking patterns like
      - Digits only
      - Specific characters
      - Structured formats (phone numbers, dates)
  - Script Example: Literal Match with Regex

```
#!/bin/bash
input="12345"
if [[ $input =~ ^[0-9]+$ ]]; then
  echo "The input contains only numbers"
fi
```

- ^[0-9]+$ explanation
  - ^
    - Start of line
  - [0-9]+
    - One or more digits
  - $
    - End of line
- Only prints the message if the pattern matches the full string

- Summary
  - Regular expressions match patterns, not exact values
  - Bash uses [[$variable =~ pattern]] for regex checks
  - Essential for
    - Input validation
    - Matching structured data
    - Smarter scripting logic

- **Test Operators**
  - -d and -f
    - *-d*
      - Checks if a directory exists
    - *-f*
      - Checks if a regular file exists
    - Used to verify filesystem objects before running actions
    - Example

```
if [ -d "myfolder" ]; then
    echo "myfolder is a directory."
fi
if [ -f "myfile.txt" ]; then
    echo "myfile.txt is a file."
fi
```

- -z and -n
  - *-z*
    - Checks if a string is empty
  - *-n*
    - Checks if a string is not empty
  - Useful for validating variables or user input
  - Example

```
name=""
if [ -z "$name" ]; then
    echo "The name is empty."
fi

name="Alice"
if [ -n "$name" ]; then
    echo "The name is not empty."
fi
```

- *!*
  - Negates the result of a test condition
  - ! -f file.txt is true when file.txt does not exist
  - Enables logic that runs only when something is not true
  - Example

```
if [ ! -f "data.txt" ]; then
    echo "data.txt does not exist."
fi
```

- Summary

- -d and -f check for directories and regular files
- -z and -n validate whether strings are empty or populated
- ! reverses any test condition for flexible logic
- All test operators return true or false and are used inside [ ]
- These allow scripts to respond to file presence, variable contents, and inverted logic conditions

- **Variables**
  - *Positional Arguments*
    - Represent values passed to a script when run from the command line
    - Named based on position
    - Example
      - $1 = first argument
      - $2 = second argument
    - Allow scripts to accept input without editing the script
    - Example

```
#!/bin/bash
if [ $1 -gt 10 ]; then
    echo "The number you entered is greater than 10"
else
    echo "The number you entered is 10 or less"
fi
```

    - Command
      - ./myscript.sh 15
    - 15 becomes $1 and is evaluated in the conditional logic
  - *Environment Variables*
    - Built-in values provided by the system or user
    - Common examples

- $USER = current username
- $HOME = user's home directory
- $SHELL = current shell

- Can be used directly in scripts to adapt to the environment
- Example

```
#!/bin/bash
if [ $USER = "root" ]; then
    echo "You are logged in as the root user"
else
    echo "You are logged in as a regular user: $USER"
fi
```

- $USER holds the current login username and is used for string comparison

- Summary
  - Variables store and use information like input, text, and system data
  - Positional arguments use $1, $2, etc., to represent user-supplied input
  - Environment variables use names like $USER, $HOME, and $SHELL to reflect system values
  - Enable scripts to react to user input and environment without modification

- **Alias and Command Management**
  - *alias*
    - Used to create a shortcut for a longer command or sequence
    - Syntax

- alias name='command'
- Simplifies repeated or commonly used commands
- Example
  - alias checkdisk='df -h'
    - Typing checkdisk will run df -h
    - Saves time and reduces typing errors

- *unalias*
  - Removes an alias that was previously set
  - Syntax
    - unalias name
  - Useful when the alias is no longer needed or interferes with default commands
  - Example
    - unalias checkdisk
      - Removes the checkdisk alias so df can be used in its original form
      - Aliases are temporary unless added to .bashrc or other config files

- *set*
  - Changes shell behavior using flags and options
  - Syntax
    - set [option]
  - Common flags
    - -e = stop script on any command failure
    - -x = print each command before executing (debugging)
    - -u = exit when using undefined variables

- -o pipefail = fail script if any command in a pipeline fails

- Example

```
#!/bin/bash
set -e
echo "Starting system update..."
sudo apt update
sudo apt upgrade -y
echo "Update complete."
```

- Ensures scripts stop on failure and behave predictably

- Summary
  - alias simplifies frequently typed commands into short labels
  - unalias removes those shortcuts from the session
  - set modifies how Bash behaves for better control and script safety
  - These tools support streamlined workflows and robust script behavior

- **Variable Management**
  - *export*
    - Makes a variable available to child processes such as subshells or external scripts
    - Syntax
      - export VARIABLE=value
    - Regular variables are not visible to child shells unless exported
    - Example

```
export LOG_LEVEL=debug
./run_backup.sh
```

      - The script run_backup.sh can access LOG_LEVEL because it is exported

- Supports cross-environment communication between parent and child shells

- *local*
  - Restricts a variable's scope to within a function
  - Syntax
    - local VARIABLE=value
  - Only valid inside functions
  - Prevents conflicts between variables in functions and the main script
  - Example

```
calculate_load() {
  local load=$(uptime | awk -F'load average: ' '{ print $2 }')
  echo "Current load average: $load"
}
calculate_load
```

    - load variable is available only within calculate_load
    - Keeps function-level variables isolated and temporary

- *unset*
  - Deletes a variable or function from the current shell session
  - Syntax
    - unset VARIABLE
  - Used to remove temporary or unneeded variables
  - Helps maintain a clean environment in scripts
  - Example

```
temp_file="report_2024.txt"
echo "Processing $temp_file"
unset temp_file
```

        - After unset, temp_file no longer exists and returns nothing if echoed
    - Summary
        - export shares variables with child processes and external scripts
        - local confines variables to function scope to avoid script-wide conflicts
        - unset deletes variables to free memory and prevent accidental reuse
        - These commands improve variable control, script clarity, and reliability

- **Return Codes**
    - *Return Codes*
        - Numeric values returned by commands after execution
        - Indicate whether the command was successful or failed
        - 0 means success
        - Non-zero means failure or error
        - Accessed using the special shell variable $?
        - Each command can have its own specific error codes
    - Using Return Codes with $?
        - Syntax
            - echo $?
        - Displays the return code of the last executed command
        - Must be used immediately after the command to get an accurate result
    - Example: Successful Command

```
ls /tmp
echo $?
```

- ls /tmp runs successfully
- echo $? outputs 0
- Confirms the command executed without error

- Example: Failed Command

```
sudo usermod -c "Test user" no_such_user
echo $?
```

- usermod fails because the user doesn't exist
- Returns error message
- echo $? outputs 2, indicating a specific failure for "user not found"

- Common Return Code Behaviors
  - Each command has its own set of non-zero return codes
  - Examples
    - rm returns 1 if file doesn't exist
    - usermod returns
      - 1 for general errors
      - 2 if user doesn't exist
      - 6 if group doesn't exist
      - 8 if user is currently logged in

- Use in Scripting
  - Return codes help scripts make decisions
  - Based on return code, scripts can
    - Retry an operation
    - Alert an administrator
    - Exit gracefully
  - Essential for automation and robust error handling

- Summary
  - Every Linux command ends with a return code (exit status)
  - 0
    - Success
  - Non-zero
    - Error or failure
  - Use echo $? to inspect the return code of the last command
  - Understanding return codes is key for diagnostics and scripting

Python Basics

Objective 4.3: *Summarize Python basics used for Linux system administration*

- **Setting up a virtual environment**
  - Overview of Virtual Environments
    - A virtual environment is an isolated workspace for a Python project
    - Keeps dependencies separate from the system and other projects
    - Prevents conflicts between different versions of packages
  - Creating and Activating a Virtual Environment
    - Command to create
      - python3 -m venv <environment-name>
    - Command to activate
      - source <environment-name>/bin/activate
    - Creates a directory containing
      - A Python interpreter
      - A place to install project-specific packages
    - Example
      - python3 -m venv myprojectenv
      - source myprojectenv/bin/activate
    - Prompt changes to show active environment
      - (myprojectenv) $
  - Managing Dependencies Inside the Environment
    - Use pip to install packages within the virtual environment
    - General syntax

        - pip install <package-name>
    - Example
        - pip install requests
    - Packages are installed only within the active virtual environment
    - Helps maintain clean and conflict-free development environments
- Summary
    - Virtual environments isolate dependencies per project
    - Creation
        - python3 -m venv <environment-name>
    - Activation
        - source <environment-name>/bin/activate
    - Package installation
        - pip install <package-name>
    - Keeps projects organized, stable, and easy to maintain

- **Built-in modules**
    - Overview of Built-in Modules
        - Built-in modules are pre-installed toolkits included with Python
        - Provide reusable code for common tasks like math, random numbers, file handling, and dates
        - No additional installation required
    - Using Built-in Modules
        - Syntax to import a module
            - import module_name
        - Common modules and their purposes

- math for mathematical operations
- random for simulations and random values
- datetime for handling dates and times
- os for file and environment management
- sys for system-level access
- statistics for basic statistical operations
- time for delays and timestamps
- json for working with JSON data
- collections for advanced data structures like Counter and defaultdict

- Example usage
  - import random
  - print(random.randint(1, 10))
    - Generates a random integer from 1 to 10 and prints it

- Discovering Built-in Module Features
  - help(module_name) displays documentation for the module
  - Example
    - import math
    - help(math)
    - Lists functions like ceil() and sqrt() and constants like pi
  - Additional discovery tools
    - dir(module_name) shows all functions and attributes
    - type() checks the type of returned objects
    - __doc__ provides brief descriptions
    - Python REPL allows for interactive testing and experimentation

- Summary
  - Built-in modules provide ready-to-use tools for common programming needs
  - Usage begins with the import statement
  - Examples include math, random, os, and datetime
  - Use discovery tools like help() and dir() to explore available functions
  - Enable efficient coding without installing external packages

- **Installing dependencies**
  - Overview of Python Dependencies
    - Dependencies are external packages or libraries that support specific tasks in a Python project
    - Managed separately per project to avoid conflicts
    - Managed using
      - pip
      - requirements.txt
      - virtual environments
  - *pip: Installing Packages*
    - pip is the primary tool to install Python packages from the Python Package Index
    - Syntax
      - pip install package-name
    - Example
      - pip install requests
        - Installs the requests library for making HTTP requests

- Automatically downloads and installs required dependencies into the active environment

- *requirements.txt: Listing and Sharing Dependencies*
  - requirements.txt stores the list of installed packages and their versions
  - Created using the command
    - pip freeze > requirements.txt
  - Example file content
    - requests==2.31.0
    - urllib3==1.26.18
  - Used for replicating the environment with the command
    - pip install -r requirements.txt
- *Dependency Isolation: Virtual Environments*
  - Virtual environments isolate package installations per project
  - Prevent conflicts between project dependencies and system Python packages
  - Syntax to create
    - python3 -m venv env-name
  - Syntax to activate
    - source env-name/bin/activate
  - All pip installs are scoped to the active virtual environment
- Summary
  - pip installs external Python packages
  - requirements.txt records and shares project dependencies
  - Virtual environments isolate dependencies for each project
  - These tools ensure clean, consistent, and reproducible Python development environments

- **Python Fundamentals**
  - Overview of Python Basics
    - *Python*
      - Is a user-friendly programming language used for automation, web development, and data analysis
    - Two key fundamentals when starting with Python are
      - Understanding versioning
      - Using indentation
  - Python Versions
    - Python 2 is deprecated; Python 3 is the current standard
    - Minor versions like 3.8, 3.10, and 3.12 include slight feature differences
    - Use the command python3 --version to check the active version
    - Multiple versions can coexist on a system
    - To use a specific version, run the associated command
    - Example
      - python3.10
      - python3.12
    - Version awareness prevents errors and confusion in development environments
  - Indentation in Python
    - Indentation is mandatory and defines code structure
    - Required to group statements like those under conditionals or loops
    - Python uses whitespace instead of braces {} to indicate code blocks
    - Example

```
x = 10
if x > 5:
    print("x is greater than 5")
```

        - Improper indentation causes syntax errors
        - Enforces readability and structure as part of Python's design philosophy
    - Summary
        - Python is simple, readable, and powerful for beginners and professionals
        - Python 3 should be used for all new projects, and version control helps avoid conflicts
        - Indentation is a core part of Python syntax, not just formatting
        - Following these basics ensures code runs correctly and remains maintainable

- **Python Data Types**
    - Overview of Data Types
        - Data types classify values so Python can process them correctly
        - Each type organizes values based on how they should be stored and used
        - Four fundamental data types in Python are integers, floating point numbers, strings, and booleans
    - *Integer*
        - Represents whole numbers without decimals
        - Examples
            - 10
            - -3
            - 42
        - Used in counting, ages, and other non-fractional calculations

- Can be used in mathematical operations such as addition, subtraction, multiplication, and division
- Assigned using simple syntax
- Example
  - apples = 10
    - Spacing around the equal sign does not affect execution

- *Floating Point*
  - Represents numbers with decimal points
  - Examples
    - 3.14
    - 0.5
    - -2.75
    - 5.0 (float, not integer)
  - Used when decimal precision is required such as in prices or measurements
  - Supports the same mathematical operations as integers
- *String*
  - Represents a sequence of characters enclosed in quotes
  - Can include letters, numbers, punctuation, and spaces
  - Examples
    - name = "Alex"
    - message = 'Hello, world!'
    - "25" (treated as text, not a number)
  - Useful for storing names, messages, file paths, and human-readable data
- *Boolean*
  - Represents logical values of either True or False

- Used to track states or control flow in a program
- Assigned directly without quotation marks
- Example
  - is_active = True
  - logged_in = False
- Recognized as keywords and used in conditional logic

- Summary
  - Integers are used for whole numbers in math operations
  - Floating points provide precision with decimal values
  - Strings handle any kind of textual content and must be in quotes
  - Booleans manage logical decisions with True or False values
  - Understanding and applying data types ensures the program behaves as expected

- **Python Structures**
  - *List*
    - Used to store a collection of items in a specific order
    - Defined using square brackets []
    - Items are separated by commas
    - Indexing starts at 0
    - Items are accessed by their index positions
    - Supports modifying, adding, and removing items using index
    - Example

```
fruits = ["apple", "banana", "cherry"]
print(fruits[1])  # Outputs "banana"
```

- *Dictionary*
  - Used to store data in key-value pairs
  - Defined using curly braces {}
  - Each key is followed by a colon and its corresponding value
  - Keys act as labels for easy data retrieval
  - Values are accessed using their keys, not index positions
  - Example

```
person = {"name": "Alice", "age": 30, "city": "New York"}
print(person["age"])  # Outputs 30
```

- Summary
  - Lists use ordered positions (indexes) to access items
  - Dictionaries use descriptive keys for direct access to values
  - Both structures are essential for organizing and managing multiple data items in Python scripts

- **Extensibility using Python modules and packages**
  - Overview
    - Python is extensible
    - Can add code to expand its functionality
    - Python supports code reuse and modularity
    - Three types of extensibility
      - Modules
      - Packages
      - Third-party packages

- *Modules*
  - A module is a Python file (.py) containing reusable code
  - Examples of built-in modules
    - math
    - random
    - datetime
  - Use modules by importing them
    - Example
      - Import math
      - Then use math.sqrt() to calculate square roots
- *Packages*
  - A package is a folder that groups multiple related modules
  - Helps organize larger projects
  - Examples from standard library
    - os
    - http
  - Use them like modules
    - Examples
      - import os
      - from http import client
  - Packages can contain many modules
- Third-party packages
  - Created by the Python community
  - Installed using pip
  - Examples
    - requests

        - pandas
        - flask
    - Import like regular modules
        - Example
            - import requests
- Summary
    - Modules
        - Single reusable Python files
    - Packages
        - Folders with multiple modules
    - Third-party packages
        - Installable tools from other developers
        - Use import to access them
        - Built-ins come with Python
        - Third-party packages require pip install

- **PEP 8 Best Practices**
    - *PEP 8 Rules*
        - PEP 8 is the official style guide for Python
        - Promotes clean, consistent, and readable code
        - Indent using 4 spaces
        - Keep lines under 79 characters
        - Add blank lines between functions
        - Use lowercase_with_underscores for variable and function names
        - Use CamelCase for class names
        - Place one space around operators (=, +, -)

- Do not include spaces directly inside parentheses, brackets, or braces
- Reference
  - https://peps.python.org/pep-0008/

- *flake8*
  - flake8 is a Python tool that checks code against PEP 8 standards
  - Installed with pip
    - pip install flake8
  - Run it on a file using
    - flake8 filename.py
  - Returns messages with line numbers and issue codes
  - Identifies problems like missing spaces, long lines, and more
- Summary
  - PEP 8 provides formatting rules to write clear and standardized Python code
  - flake8 helps enforce PEP 8 rules by automatically scanning and reporting style issues
  - Using both ensures better code readability and maintainability across teams and projects

Implement Git

Objective 4.4: *Given a scenario, implement version control using Git*

- **Git Basics**
  - *git init*
    - Initializes a new Git repository in the current directory
    - Creates a hidden .git folder containing internal Git data
    - Enables tracking of file changes in the directory
    - Example command
      - git init
    - Example output
      - Initialized empty Git repository in /home/user/myproject/.git/
  - *git config*
    - Sets user information and preferences for Git
    - Used to record user identity in commit history
    - Common syntax
      - git config --global user.name "Your Name"
      - git config --global user.email "you@example.com"
    - Example
      - git config --global user.name "Alice Smith"
      - git config --global user.email "alice@example.com"
        - --global makes the setting apply to all repositories on the system
  - *.gitignore*

        - Specifies files or patterns Git should ignore
        - Prevents unnecessary or sensitive files from being tracked
        - File must be created in the root of the Git repository
        - Example command to create the file
            - touch .gitignore
        - Example content of .gitignore file
            - *.log
        - Example command to verify what Git is tracking
            - git status
    - Summary
        - git init sets up a project for version control by creating a repository
        - git config defines user identity used in commit records
        - .gitignore tells Git which files to exclude from tracking
        - Together, these provide a foundation for using Git to manage changes and collaborate on code

- **Git Changes**
    - *git add*
        - Stages changes for the next commit
        - Prepares selected files for inclusion in version history
        - Example syntax
            - git add <filename>
            - git add .
        - Example usage
            - git add backup.sh

- Adds backup.sh to the staging area for inclusion in the next commit
- Allows selective staging of files for better collaboration and version control

- *git commit*
  - Saves staged changes to the Git project history
  - Records a snapshot with a descriptive message
  - Example syntax
    - git commit -m "commit message"
  - Example usage
    - git commit -m "Updated backup script to include daily rotation"
  - Example output

```
[main 5f3e123] Updated backup script to include daily rotation
1 file changed, 5 insertions(+), 2 deletions(-)
```

    - Includes branch name, commit ID, commit message, and summary of file changes

- *git log*
  - Displays a list of all past commits
  - Shows commit ID, author, date, and commit message
  - Example usage
    - git log
  - Example output

```
commit 5f3e123abc456def7890abc123456def7890abcd
Author: Admin User <admin@example.com>
Date:   Mon May 20 10:15:43 2025 -0500

    Updated backup script to include daily rotation
```

        - Useful for reviewing project history and tracking contributions
    - *git diff*
        - Displays differences between file versions
        - Can compare working directory to staging area or between commits
        - Example usage

```
git diff
git diff <commit1> <commit2>
git diff backup.sh
```

        - Example output

```
-OLD_BACKUP_DIR="/var/backups"
+OLD_BACKUP_DIR="/mnt/backups"
```

            - Shows removed lines with - and added lines with +
            - Helps verify changes before committing
    - Summary
        - git add stages files for inclusion in the next commit
        - git commit records staged changes into the project history
        - git log displays the history of commits with detailed information
        - git diff highlights differences between file versions or commit snapshots
        - These commands form the foundation for effective team collaboration and version control using Git

- **Git Versions**
    - Overview

- Git enables multiple developers to work simultaneously without conflict
- Branches allow for separate lines of development
- Key versioning commands include git branch, git checkout, git merge, and git rebase
- Squash merging and rebasing help clean up project history

- *git branch*
  - Used to create and list branches in a repository
  - Branches enable isolated development without affecting the main line
  - Syntax
    - git branch <branch-name>
  - Example
    - git branch feature-scheduled-backups
      - Shows list of branches with git branch
      - Current branch is marked with *
- *git checkout*
  - Switches working directory to another branch
  - Allows context-switching between different lines of development
  - Syntax
    - git checkout <branch-name>
  - Example
    - git checkout feature-scheduled-backups
  - Output
    - Switched to branch 'feature-scheduled-backups'
      - Changes from this point are saved to the checked-out branch

- *git merge*
  - Integrates changes from one branch into another
  - Must be run from the branch receiving the changes
  - Syntax
    - git merge <branch-name>
  - Example
    - git merge feature-scheduled-backups
  - Output might include
    - Updating 9a1c2d3..e4f5g6h

      Fast-forward

      backup.sh | 12 ++++++++++--
  - Squash merges combine multiple commits into a single commit
  - Syntax
    - git merge --squash <branch-name>
  - Example
    - git merge --squash feature-scheduled-backups
  - Prompts user to enter a commit message summarizing all changes
- *git rebase*
  - Reapplies commits from one branch onto another
  - Helps create a clean, linear commit history
  - Syntax
    - git rebase <target-branch>
  - Example
    - git rebase main
  - Output
    - First, rewinding head to replay your work on top of it...

Applying: Added scheduled task for backup

Applying: Fixed logging in backup script

- Rebase rewrites commit history
- Should be used with caution on shared branches to avoid conflicts

- Summary
  - git branch creates and lists independent lines of development
  - git checkout switches between those lines
  - git merge brings changes together, with optional squash for cleaner commits
  - git rebase reorders commit history to appear linear and sequential
  - Merges preserve branch history, rebases rewrite it
  - Squash merges and rebases help maintain a clean and readable project history

- **Git Repositories**
  - *Git Repository*
    - A Git repository is a project folder that remembers every change made
    - Allows revisiting earlier versions and safe collaboration
    - Exists in two places
      - Remote copy (e.g., GitHub)
      - Local copy (your computer)
    - Most Git commands move updates between local and remote
  - Git Command Cycle
    - *git clone*

- Copies the entire remote repository and history to a new local folder
- *git fetch*
  - Checks remote updates without changing local files
- *git pull*
  - Brings remote commits into working files via fetch + merge or rebase
- *git push*
  - Uploads local commits to the remote server

- *Git Clone*
  - git clone <repository-url> [directory]
  - Downloads files and history from remote and creates a local folder
  - Sets up remote shortcut called origin
  - Example
    - git clone https://github.com/octo-org/hello-world.git hello-world
  - Result
    - Cloning into 'hello-world'..., remote: Enumerating objects: 42, done., and a short progress meter before finishing with Checking connectivity... done.
      - Creates hello-world directory
      - Connects origin to GitHub repo
- *Git Fetch*
  - git fetch origin main
  - Contacts origin and checks for new commits on the main branch
  - Updates hidden remote-tracking branches like origin/main
  - Does not change local working directory or branch

        - Allows preview of incoming changes before merging
    - *Git Pull*
        - git pull origin main
        - Performs a fetch and then merges or rebases the commits
        - Updates local branch and files to match the server
        - Helps reduce merge conflicts
        - Keeps local environment up to date
    - *Git Push*
        - git push origin main
        - Sends local commits to the remote repository
        - Updates remote branch pointer
        - Makes commits visible to teammates and automation tools
        - Example
            - git push origin main
    - Summary
        - Git repositories track every change
        - Exist on both local machine and remote server
        - Typical workflow
            - clone → fetch → pull → push
        - Keeps code current and avoids merge conflicts

- **Managing Git Workflows**
    - Overview
        - Git workflows help manage project history efficiently
        - Key tools include git tag, git stash, and git reset

- These tools allow milestone tracking, temporary change storage, and history manipulation

- *git tag*
  - Tags assign a permanent label to a specific commit
  - Useful for marking release points or stable versions
  - Syntax
    - git tag [-a] <tag-name> [commit-id]
      - -a adds an annotation message
  - Example
    - git tag -a v1.0 -m "First stable release"
  - Tags point to exact commits, independent of branch changes
  - Tags are immutable and can be referenced any time
  - HEAD is the current commit checked out, often used as the default for tagging
- *git stash*
  - Stashes temporarily store uncommitted changes
  - Clears the working directory and staging area for task switching
  - Syntax
    - git stash push [-m "message"]
  - Example
    - git stash push -m "WIP: new login form"
  - Creates a hidden stack entry with a message
  - Working directory becomes clean after stash
  - Restore changes using git stash pop or git stash apply
  - Prevents committing partial work to the wrong branch

- *git reset*
  - Resets move the branch pointer to a previous commit
  - Can adjust the staging area and working directory depending on the flag used
  - Syntax
    - git reset [--soft|--mixed|--hard] <commit-id>
  - Example
    - git reset --hard HEAD~1
      - --soft keeps staging and working directory unchanged
      - --mixed resets staging but not working directory
      - --hard resets everything (branch, staging, working files)
      - HEAD~1 means one commit before the current HEAD
      - Use –hard cautiously as it discards local changes permanently
- Summary
  - git tag creates fixed reference points in the repository for releases or milestones
  - git stash temporarily saves work-in-progress to allow context switching
  - git reset edits or discards recent history by shifting branch pointers
  - Together, these tools ensure history is organized, reversible, and accurate

Use of Artificial Intelligence

Objective 4.5: *Summarize best practices and responsible uses of artificial intelligence (AI)*

- **Code Generation Use Cases**
  - Overview
    - AI-driven code generation turns plain English into working code
    - Saves time and reduces syntax errors
    - Applies to regular expressions, full code, and Infrastructure as Code
  - *Generation of Regular Expressions*
    - AI can generate regex patterns to match formats like email or phone numbers
    - Example prompt
      - "Write a regex in JavaScript that matches U.S. phone numbers with or without dashes"
      - "Create a Python-compatible regex to validate email addresses, allowing periods and hyphens in the username"
    - Regex can be tested with sample strings or online testers
    - Providing clear requirements and valid/invalid examples improves accuracy
  - *Generation of Code*
    - AI can generate complete code blocks from prompts
    - Prompt should include
      - Programming language
      - Function or class name

        - Inputs and outputs
        - Edge cases
    - Example prompt
        - “Write a Python function called calculate_tax that takes a subtotal and returns a 7% sales tax, rounding to two decimal places”
        - “Give me a JavaScript module that fetches JSON from a URL and logs any 404 errors”
    - Code includes comments and basic error handling when requested
    - Result can be copied, pasted, tested, and adjusted in a local environment
- *Generation of Infrastructure as Code*
    - AI can generate declarative files for cloud infrastructure
    - YAML is commonly used for platforms like Kubernetes and AWS CloudFormation
    - Example prompt
        - “Create a Kubernetes deployment YAML that runs three replicas of an Nginx container, exposing port 80”
        - “Generate a CloudFormation YAML template for an S3 bucket with versioning enabled and a public-read policy”
    - Output should be validated using a cloud provider’s validation tools
    - Generated YAML aligns with intended architecture
- Summary
    - AI can generate functional regex, code, or infrastructure templates from prompts
    - Enables testing and implementation without memorizing syntax
    - Supports faster and more reliable development workflows

- **Code Quality Use Cases**
  - Overview
    - Code quality is critical for readability, maintainability, and efficiency
    - AI has automated and enhanced key areas of code quality management
    - Main use cases include code linting, code documentation, and code optimization
  - *Code Linting*
    - Code linting involves detecting and fixing code errors and inconsistencies
    - AI linters automatically identify common mistakes and enforce style rules
    - Reduces bugs and ensures consistent code across teams
    - Learns from large datasets and can predict subtle issues
    - Example
      - AI linter used by a Linux administrator to catch syntax errors and security issues in Bash scripts before deployment
  - *Documenting Code and Creating Documentation*
    - AI tools can automatically generate clear and accurate code documentation
    - Reads code and produces human-readable summaries and comments
    - Enhances collaboration and reduces onboarding time for new developers
    - Saves hours of manual documentation work
    - Example
      - AI-generated documentation used to describe custom monitoring scripts, helping other team members understand and troubleshoot them
  - *Code Optimization*
    - AI-driven optimization improves code performance and efficiency

        - Identifies bottlenecks, suggests or applies improvements
        - Reduces execution time and resource consumption
        - Continuously adapts and learns better optimization strategies over time
        - Example
            - AI tool used to optimize system backup scripts, making them faster and less resource-intensive on production servers
    - Summary
        - AI improves code quality through automated linting, documentation, and optimization
        - Linting reduces bugs and enforces standards with minimal manual review
        - Documentation becomes faster and more accessible for all team members
        - Optimization enhances performance, scalability, and resource usage
        - AI tools continue to evolve, providing ongoing benefits with minimal user intervention

- **Security and Compliance Use Cases**
    - Overview
        - Security and compliance are critical in software development and system administration
        - AI simplifies and automates these tasks for improved speed and accuracy
        - Focus areas include security review and compliance improvement recommendations
    - Security Review
        - Traditional reviews involved manual log and configuration checks

        - AI automates security checks and vulnerability detection
        - Quickly identifies risks and detects unusual system behavior
        - Reduces time spent on manual inspections
        - Enhances incident response speed and reduces breach risk
        - Example
            - AI tool used by an enterprise Linux administrator to detect misconfigured permissions or suspicious logins automatically
    - Recommendations for How to Improve Compliance
        - Traditional compliance involved audits, documentation reviews, and manual adjustments
        - AI analyzes large datasets and configurations for regulatory alignment
        - Offers clear, actionable suggestions to meet compliance standards
        - Reduces administrative errors and increases audit readiness
        - Example
            - AI tool used to check server configurations and recommend changes for regulatory compliance
    - Summary
        - AI automates and enhances system security reviews
        - AI identifies vulnerabilities and abnormal activities before they escalate
        - AI offers tailored suggestions for improving compliance with regulations and policies
        - These tools reduce manual effort and improve overall security posture

- **AI Best Practices**
    - Overview
        - AI supports tasks like development and data analysis

- Responsible usage requires best practices to ensure accuracy and compliance
- Focus areas include verifying output, avoiding copy/paste without review, and adhering to corporate policy

- Verifying Output
  - Always double-check AI-generated results for accuracy
  - AI may generate incorrect or misleading content called hallucinations
  - Review output by testing scripts in controlled environments
  - Check system responses, logs, and results to catch errors early
  - Prevent misconfigurations and functional issues in production
  - Example
    - Run AI-generated configurations in a test environment before production deployment
- Avoiding Copy/Paste Without Review and Quality Assurance
  - Blindly copying AI-generated code can introduce security flaws or bugs
  - Subtle errors may go unnoticed without careful review
  - QA practices like peer reviews and automated tests help validate output
  - Ensures safe and reliable implementation
  - Example
    - An unreviewed AI-generated security policy may open vulnerabilities across enterprise Linux systems
- Adhering to Corporate Policy
  - Follow organizational guidelines for AI usage
  - Policies often cover data privacy, AI output documentation, and auditability
  - Maintain clear records of AI-generated implementations

        - Use logging, monitoring, and periodic audits to ensure compliance
        - Example
            - Log all AI-assisted changes, document their purpose, and ensure proper access control
    - Summary
        - AI helps automate tasks but requires validation to prevent risks
        - Always verify AI output to catch hallucinations or misconfigurations
        - Avoid direct copy/paste without review and use thorough QA checks
        - Follow company AI policies to ensure transparency, traceability, and compliance

- **AI Data Governance**
    - Overview
        - AI data governance ensures responsible, secure, and ethical handling of data
        - Key focus areas include security during LLM training and consistent human review of AI outputs
        - Governance practices reduce the risk of sensitive data exposure and maintain reliability in AI-generated results
    - Security of LLM Training
        - *Large Language Models (LLMs)*
            - Are trained on massive datasets, often including sensitive information
        - Improper data handling can lead to unintentional exposure of passwords, IPs, or proprietary business content

- Example
  - An LLM trained on internal server documentation may accidentally memorize and regurgitate private credentials if not secured
- Best practices include
  - Encryption of training data
  - Role-based access control
  - Routine audits to verify no sensitive content is exposed
- Prevents data leakage during AI model development and deployment

- Human Review
  - AI can generate incorrect or misleading content if left unchecked
  - Human oversight is essential to ensure accuracy and appropriateness of AI-generated outputs
  - Example
    - A misconfigured AI-generated script deployed across servers could cause widespread outages or security breaches if not reviewed
  - Review process includes
    - Verifying AI-generated commands
    - Testing outputs in staging environments
    - Ensuring recommendations comply with organizational standards
  - Combines AI's speed with human judgment for safer implementations
- Summary
  - AI data governance ensures secure, ethical use of information in AI workflows
  - Secure training of LLMs protects against unintentional disclosure of sensitive or internal data

        - Human review provides oversight, prevents faulty automation, and enforces compliance
        - Proper governance maintains trust, reliability, and safety in AI-supported environments

- **AI Local Models**
    - Overview
        - AI local models are deployed within an organization's infrastructure
        - They can be classified as private or public based on data sensitivity and access permissions
        - Each type serves different use cases and has distinct security implications
    - *Private Local Models*
        - Hosted and accessed entirely within the organization
        - Trained on sensitive or proprietary data
        - Restricted to internal users or systems
        - Example
            - Model trained on internal security event logs to detect Linux anomalies
        - Common use cases
            - Automating internal workflows
            - Generating change summaries from helpdesk tickets
        - Security measures
            - Encryption of training data
            - Role-based access control
            - Audit logging of model access and actions

- *Public Local Models*
  - Hosted by the organization but accessible to external users
  - Trained on non-sensitive or open-source data
  - Intended for broad use such as clients, developers, or public APIs
  - Example
    - AI assistant for troubleshooting common Linux errors using open-source data
  - Common use cases
    - API for generating script templates
    - External developer support tools
  - Security measures
    - Rate limiting
    - Prompt injection protection
    - Access token authentication
    - Monitoring for abuse or repeated query attacks
- Summary
  - Private models enable secure, internal AI usage on sensitive datasets
  - Public models extend AI services externally while maintaining internal control
  - Both types require proper provisioning, deployment, and protective measures
  - Enterprise administrators must enforce strong security for both, tailored to exposure level and data sensitivity

- **Prompt Engineering**
    - *Prompt Engineering*
        - Prompt engineering is the skill of crafting precise inputs (prompts) for AI systems
        - Clear prompts guide AI to generate accurate and useful responses
        - Focuses on three key areas
            - Safe interactions
            - Optimizing prompts
            - Automation
    - *Safe Interactions*
        - AI generates responses based on patterns, not understanding
        - AI outputs must be validated before applying them in real systems
        - Built-in guardrails prevent dangerous outputs like rm -rf /
        - User-applied guardrails include
            - Testing commands in sandboxes, containers, or virtual machines
            - Avoiding direct execution on live systems
            - Reviewing commands for red flags like non-existent packages or incorrect syntax
        - Examples of AI improvisation
            - Fake package name
                - apt-get install net-tools-plus
                    - Misformatted cron job with incorrect number of fields
    - *Optimizing Prompts*
        - Optimized prompts are clear, focused, and reduce ambiguity
        - Dense prompts may cause hallucinations or skipped steps

- Example of overly dense prompt
  - "Explain how to configure a static IP address in Linux, including differences between Ubuntu and Rocky Linux, show both GUI and CLI methods, and give troubleshooting steps if it fails to work after reboot"
- Better approach: break into smaller parts
  - "How do I configure a static IP address using the command line in Ubuntu 22.04?"
    - Improves reliability and precision of AI responses

- *Automation*
  - Automation applies prompt engineering to perform repeatable tasks
  - Example use cases
    - Automating system documentation
    - Generating user management scripts
    - Summarizing system logs or error messages
  - Different models serve different needs
    - ChatGPT
      - Conversational, context-rich
    - Claude
      - Effective for summarization
    - Mistral/LLaMA
      - Suitable for self-hosted or air-gapped environments
  - Trade-offs include access type, performance, and cost
  - Automation enhances efficiency without replacing critical thinking
- Summary
  - Guardrails protect users but must be paired with user caution

- Poor prompts are vague, dense, and overambitious
- Effective prompts are specific, build logically, and are testable
- Prompt engineering improves productivity, safety, and clarity when working with AI for technical tasks like scripting and troubleshooting

Monitoring Concepts and Configurations

Objective 5.1: *Summarize monitoring concepts and configurations in a Linux system*

- **Service Monitoring**
    - Service Monitoring Concepts
        - Service Level Indicators (SLIs)
        - Service Level Objectives (SLOs)
        - Service Level Agreements (SLAs)
    - *Service Level Indicators (SLIs)*
        - SLIs measure how well a service is performing
        - Common SLIs include uptime, response time, and error rate
        - Example tools for measuring SLIs
            - ping
            - uptime
            - curl
            - grep
            - awk
            - journalctl
        - SLIs provide raw data to detect issues and assess service health
    - *Service Level Objectives (SLOs)*
        - SLOs define target goals for specific SLIs
        - Represent the threshold of acceptable service performance
        - Example
            - SLI = uptime

        - SLO = 99.9% uptime each month
    - SLOs are set through collaboration between technical and business teams
    - Monitoring tools use SLOs to trigger alerts when performance degrades
  - *Service Level Agreements (SLAs)*
    - SLAs are formal contracts that include SLIs and SLOs
    - Define the expected service level between provider and user
    - Can apply to external vendors or internal departments
    - Must specify consequences if performance targets are missed
    - Examples
        - Service credits
        - Financial penalties
    - Linux administrators help ensure SLAs are technically feasible and maintained
  - Summary
    - SLIs provide measurable service performance data
    - SLOs set performance expectations based on SLIs
    - SLAs formalize expectations and consequences for underperformance
    - Linux administrators are responsible for collecting SLIs, informing SLOs, and ensuring SLAs are upheld

- **Network Monitoring**
  - Network Monitoring Concepts
    - SNMP (Simple Network Management Protocol)
    - MIB (Management Information Base)
    - OID (Object Identifier)

- Traps
- Agent-based Monitoring
- Agentless Monitoring

- *SNMP (Simple Network Management Protocol)*
  - SNMP monitors routers, switches, servers, and other network devices
  - Devices report performance and status data to a central system
  - MIB defines what can be monitored on a device
  - Example
    - CPU load
    - Memory usage
    - Interface status
  - OID is a unique identifier for retrieving MIB data
  - SNMP can operate in two ways
    - Regular polling
    - Automatic alerts (Traps)
  - SNMP Traps notify administrators of real-time issues like failures or outages
  - SNMP tools used in Linux
    - snmpwalk
    - snmpget
- Agent vs Agentless Monitoring
  - *Agent-based Monitoring*
    - Relies on agents installed on devices
    - SNMP is an example of agent-based monitoring
    - Agents collect and send structured data
    - Often pre-installed or integrated into operating systems

- *Agentless Monitoring*
  - Uses protocols like SSH, HTTP, or APIs
  - Requires no software installation on the monitored system
  - Examples include
  - SSH scripts for Linux
  - WMI for Windows
- Agent-based advantages
  - More detail and control
  - Better security and structured data
- Agentless advantages
  - Faster deployment
  - Reduced maintenance overhead
- Choose based on
  - Network size
  - System diversity
  - Required data depth
  - Security policy

- Summary
  - SNMP provides structured, centralized monitoring through MIBs and OIDs
  - Traps send real-time alerts when issues occur
  - Agent-based monitoring installs software on devices for detailed data
  - Agentless monitoring uses remote access and APIs for quick setup
  - Choose method based on your environment's needs and policies

- **Event-driven Data Collection**
    - Event-driven Data Collection Overview
        - Enables systems to detect and respond to issues in real time
        - Improves responsiveness and reliability over scheduled checks or manual monitoring
        - Main methods include
            - Health checks
            - Webhooks
            - Log aggregation
    - *Health Checks*
        - Automated tests verifying if services are running and responding correctly
        - Can be run on a schedule or triggered by tools or scripts
        - Used by tools like Kubernetes and load balancers to make decisions
        - Example commands
            - systemctl is-active ssh
                - Confirms a systemd service is running
            - curl -I http://localhost
                - Checks for a 200 OK response from a web service
        - Helps determine if a service needs restarting or traffic redirection
    - *Webhooks*
        - Notify systems immediately when events occur
        - Used for real-time integrations between services
        - Examples
            - CI system sends webhook when a build fails
            - Security tool sends alert to Slack when a threat is detected

        - Linux admins can use webhooks to alert dashboards or trigger automation
    - *Log Aggregation*
        - Collects logs from multiple sources into a centralized location
        - Crucial in large environments where checking each log manually isn't feasible
        - Tools like SIEM ingest logs from servers, firewalls, applications, etc
        - Enables
            - Event correlation
            - Pattern detection
            - Alert triggering
            - Troubleshooting and incident investigation
            - Security policy compliance
    - Summary
        - Event-driven data collection improves system management by reacting to events instantly
        - Health checks validate service status and trigger corrective actions
        - Webhooks provide instant notifications of critical events
        - Log aggregation supports centralized troubleshooting, analysis, and monitoring across systems

- **Event Management**
    - Event Management Concepts
        - Logging
        - Events

- Thresholds
- *Logging*
  - Logging is the foundation for event management
  - Logs record system activity across devices
  - Common Linux log files
    - /var/log/syslog
    - /var/log/auth.log
    - /var/log/dmesg
  - Logs track authentication, kernel activity, and applications
  - Enterprise environments generate massive volumes of logs
  - Logs are forwarded to centralized logging systems
    - rsyslog
    - journald
  - Logs are analyzed through SIEM (Security Information and Event Management) platforms
    - Example
      - Splunk
      - Wazuh
  - Centralized logging enables correlation, retention, and visibility
- *Events*
  - Events are generated from patterns in log data
  - Events indicate activity that may require action
    - Failed logins
    - Privilege escalation
    - Service shutdown
  - SIEM tools apply rules to log data in real time

- Linux logs like /var/log/auth.log or auditd logs are parsed
- Example Wazuh rule for SSH brute-force detection

```
<rule id="5710" level="10">
  <if_sid>5712</if_sid>
  <same_source_ip />
  <frequency>6</frequency>
  <timeframe>60</timeframe>
  <description>SSH brute force attempt detected from same IP</description>
</rule>
```

  - Rule watches for 6+ failed SSH attempts within 60 seconds from the same IP
  - Event is raised if rule is triggered, enabling
    - IP blocking
    - Notification
    - Automated scripts

- *Thresholds*
  - Thresholds determine when events trigger alerts or actions
  - Provide logic to separate normal from abnormal behavior
  - Example
    - 1 failed login = normal
    - 6 failed logins in 1 minute = brute-force attempt
  - Thresholds apply to performance metrics
    - memory usage
    - CPU load
  - Proper thresholding avoids alert fatigue while catching real issues
- Summary
  - Logging captures all system activity and is essential for event tracking
  - Events are generated from log analysis when rules are matched

        - Thresholds define conditions for alerting or action
        - Together, logging, events, and thresholds form a complete event management strategy for enterprise Linux environments

- **Alerting and Notifications**
    - Alerting and Notifications Concepts
        - Notifications
        - Alerts
    - *Notifications*
        - Notifications inform administrators of conditions requiring attention
        - Delivered automatically when issues or status changes occur
        - Delivery methods
            - Email
            - SMS
            - Desktop pop-up
            - Integrated ticketing systems
        - Notifications include
            - Timestamp
            - Hostname
            - Affected component
            - Severity
            - Issue description
        - Severity levels
            - Informational
                - No immediate action needed

- Warning
  - Possible problem developing
- Critical
  - Needs immediate attention

- Example notifications
  - Informational
    - Successful user login
  - Warning
    - Disk usage at 80%
  - Critical
    - Web service stopped responding

- *Alerts*
  - Alerts are internal triggers activated by monitoring thresholds
  - Generated based on predefined system metrics
    - CPU load
    - Memory usage
    - Disk space
    - Network status
    - Service availability
  - Can include security-related conditions
    - Failed logins
    - Permission changes
  - Severity levels categorize alert impact
    - Informational
    - Warning
    - Critical

- Alert refinement to reduce noise and false positives
  - Evaluation rules
  - Persistence thresholds
  - Multiple-event correlation
- Example alert behavior
  - CPU hits 90%
    - No alert
  - CPU remains at 90% for 5 minutes
    - Critical alert triggered
- Once valid, alerts pass data to notification systems for administrator response

- Summary
  - Alerts are triggered when performance or security thresholds are met
  - Notifications deliver alert info to administrators via various channels
  - Severity levels and formatted messages help prioritize response
  - Evaluation rules help filter out false positives
  - Together, alerts and notifications ensure system health is monitored and issues are addressed quickly

Troubleshooting Hardware, Storage, and Linux OS

Objective 5.2: *Given a scenario, analyze and troubleshoot hardware, storage, and Linux OS issues*

- **Boot Issues**
  - Common Linux Boot Failures
    - Server not turning on
    - GRUB misconfiguration
    - Kernel corruption
    - Missing or disabled drivers
    - Kernel panic
  - *Structured Troubleshooting Approach*
    - Identify the problem
    - Determine the scope
    - Establish a theory of probable cause
    - Test the theory
    - Establish a plan of action
    - Implement the solution
    - Verify full functionality
    - Implement preventive measures
    - Perform root cause analysis
  - *Server Not Turning On*
    - Problem symptoms
      - No power lights
      - No fans

        - No console output
    - Scope assessment
        - Check if other servers are affected
    - Probable causes
        - Failed power distribution unit (PDU)
        - Failed server power supply unit (PSU)
    - Testing actions
        - Check rack PDU status
        - Swap in known-good power cable
        - Test outlet with another device
    - Solutions
        - Inspect PSU
        - Reseat connectors
        - Replace PSU
    - Verification
        - Confirm system powers on
        - Access BIOS or BMC
    - Preventive measures
        - Label cables
        - Schedule PSU checks
    - Root cause examples
        - Failed PSU
        - Tripped breaker
- *GRUB Misconfiguration*
    - Problem symptoms
        - GRUB rescue prompt

- “file not found” error

- Scope assessment
  - Check if one or multiple kernels fail
- Probable causes
  - Incorrect root= UUID
  - Deleted initrd entry in /etc/default/grub
- Testing actions
  - Probe partitions using GRUB command line
  - Verify presence of kernel/initramfs files
- Solutions
  - Boot from rescue ISO
  - Mount root filesystem
  - Fix UUID/kernel path
- Regenerate GRUB config
  - RHEL
    - grub2-mkconfig -o /boot/grub2/grub.cfg
  - Debian
    - update-grub
- Verification
  - Reboot and check kernel loads cleanly
- Preventive measures
  - Validate GRUB changes in test
  - Backup grub.cfg
- Root cause examples
  - Rushed update
  - Lack of peer review

- *Kernel Corruption*
  - Problem symptoms
    - “bad magic number”
    - “kernel image corrupt”
  - Scope assessment
    - Test other kernel versions in GRUB menu
  - Probable causes
    - Interrupted kernel update
    - /boot partition disk errors
  - Testing actions
    - Boot older kernel
    - Verify checksums on kernel files
  - Solutions
    - Reinstall corrupted kernel package
  - Verification
    - Confirm new kernel boots
  - Preventive measures
    - Monitor disk health (e.g. smartctl)
    - Ensure updates complete successfully
  - Root cause examples
    - Disk failure
    - Power loss during update
- *Missing or Disabled Drivers*
  - Problem symptoms
    - System hangs at boot
    - initramfs shell shows “VFS: Cannot open root device”

- Scope assessment
  - Identify missing devices in /dev or /sys
- Probable causes
  - Missing driver in initramfs
  - Blacklisted driver
- Testing actions
  - Check initramfs using lsinitrd or dracut --list-modules
- Solutions
  - Rebuild initramfs with required modules
- Verification
  - Confirm root filesystem detected at boot
- Preventive measures
  - Document driver dependencies
  - Automate initramfs rebuilds after kernel updates
- Root cause examples
  - Configuration error
  - Incomplete package build

- *Kernel Panic*
  - Problem symptoms
    - Message on console
      - “Kernel panic – not syncing”
  - Scope assessment
    - Determine if panic happens consistently or after changes
  - Probable causes
    - Incompatible kernel module
    - Memory failure

        - Hardware change
    - Testing actions
        - Boot into previous kernel
        - Run memtest86+
        - Disable modules via kernel boot line
    - Solutions
        - Remove/update modules
        - Roll back kernel
        - Replace faulty RAM
    - Verification
        - Reboot and ensure stable system
    - Preventive measures
        - Maintain kernel testing process
        - Monitor hardware
        - Document tested modules
    - Root cause examples
        - Faulty driver
        - Bad memory
        - Untested hardware
    - Windows comparison
        - Kernel panic equals Blue Screen of Death (BSOD)
- Summary
    - Boot issues can stem from hardware failure, configuration errors, or corrupted software
    - Each failure mode has a specific diagnostic process and recovery step

- Preventive maintenance and documentation help reduce future boot problems
- A consistent troubleshooting workflow ensures reliable and repeatable results

- **Filesystem Issues**
  - Overview
    - Filesystems organize data into directories and files accessible by the OS
    - Five major filesystem issues covered in this lesson
      - Filesystem will not mount
      - Partition is not writable
      - OS filesystem is full
      - Inode exhaustion
      - Quota issues
  - *Filesystem Will Not Mount*
    - Symptom
      - mount command returns errors or data becomes inaccessible
    - Error messages may include
      - Unknown filesystem type
      - Mount
        - Wrong fs type
      - Superblock corrupt
    - Recovery steps
      - Boot into rescue mode or unmount stale references
      - Run fsck and repair the superblock if needed

- Correct any misconfigured /etc/fstab entries (UUID or device path)
- Test the mount manually before updating fstab
- Confirm read/write access and update monitoring tools

- *Partition Is Not Writable*
  - Symptom
    - "permission denied" or failure to save files
  - Check if the filesystem is mounted as read-only
    - Look for ro flag in mount or /proc/mounts
    - Use journalctl or dmesg to confirm auto-remounts due to errors
  - Resolution
    - Unmount the partition and run fsck
    - Remount with correct read-write permissions
    - If permission-based, check ownership and ACLs
    - Use chmod or chown to correct access
    - Update config scripts to ensure proper mounting at boot
- *OS Filesystem Is Full*
  - Symptom
    - "No space left on device" and failing services
  - Confirm 100% usage of root or target partition
  - Fix options
    - Truncate or rotate log files
    - Clean up core dumps and orphaned Docker images
    - Archive old data to other storage
    - Resize LVM volume and filesystem with help from storage teams
  - Prevention
    - Enable monitoring and alerts for low disk space

- *Inode Exhaustion*
  - Symptom
    - "No space left on device" despite free space in df -h
  - Check df -i to confirm inode usage is at 100%
  - Typical in systems storing millions of small files
  - Remediation
    - Clean up directories with excessive small files
    - If necessary, create a new filesystem with more inodes
    - Migrate data to the new filesystem
  - Prevention
    - Update cleanup policies or add auto-delete scripts
- *Quota Issues*
  - Symptom
    - “Disk quota exceeded” for specific users or groups
  - Check quota usage with
    - repquota -a
    - quota -u username
  - Fixes
    - Raise soft/hard quota limits if usage is valid
    - Remove unneeded files if overuse or orphaned data is found
  - Prevention
    - Set warnings for 80% usage to avoid future issues
- Summary
  - Filesystem issues must be resolved quickly to prevent production disruptions
  - Key problems include

- Unmounted filesystems make all data inaccessible
- Mounted but read-only partitions require remounting or permission fixes
- Full filesystems or exhausted inodes block new data writes
- Quotas restrict usage even with available space

- Proactive monitoring, timely cleanup, and system audits are essential for prevention

- **Process Issues**
  - Overview
    - Linux systems run many processes simultaneously
    - Four main process issues discussed in this lesson
      - Unresponsive processes
      - Killed processes
      - Segmentation faults
      - Memory leaks
  - *Unresponsive Processes*
    - Symptom
      - Process does not respond to input or system events
    - Detected using
      - top or ps showing “D” (uninterruptible sleep)
      - High CPU usage with no output
      - strace to watch for stuck system calls
    - Remediation
      - Attempt SIGTERM for graceful shutdown

- Escalate to SIGKILL if needed
- Investigate root cause with logs (journalctl, application logs)
- Apply configuration changes or updates to prevent recurrence

- *Killed Processes*
  - Symptom
    - Process stops unexpectedly without console message
  - Detected using
    - journalctl or dmesg showing “Killed process” or “oom_reaper”
  - Causes
    - Out-Of-Memory (OOM) killer
    - Manual administrator action
  - Resolution
    - Check RAM usage with free -m or dashboard
    - Adjust service memory limits with MemoryLimit= or ulimit
    - Review audit logs to trace manual termination
    - Update documentation and permissions to avoid accidental kills
- *Segmentation Fault*
  - Symptom
    - Process crashes with “Segmentation fault (core dumped)”
  - Occurs when accessing invalid memory
  - Seen in logs or journalctl
  - Resolution
    - Configure system to retain core dumps
    - Use gdb to analyze core files and trace bug
    - Update or reinstall faulty packages
    - Apply patch or config change, retest to confirm stability

- *Memory Leaks*
  - Symptom
    - Process memory usage (RES) increases continuously
  - May lead to
    - System swap usage
    - Performance degradation
  - Detection
    - Monitor RES value in process tools like top or ps
  - Fix
    - Analyze app logs or diagnostics for retained memory
    - Identify and fix code paths that don't release memory
    - Schedule periodic restarts or allocate more RAM temporarily
    - Track RES post-fix over time to ensure stability
- Summary
  - Process issues reduce system stability and performance
  - Key types
    - Unresponsive processes
      - Hang without responding to input
    - Killed processes
      - Terminated by OOM killer or admins
    - Segmentation faults
      - Invalid memory access crashes
    - Memory leaks
      - RAM usage grows endlessly
  - Diagnosed using top, ps, journalctl, gdb, strace, and memory tools
  - Resolved with signals, debugging, patching, or restart policies

- **System Issues**
    - Overview
        - Linux servers power critical applications and may face these four system-level issues
            - Device failure
            - Data corruption issues
            - Systemd unit failures
            - Server inaccessibility
    - *Device Failure*
        - Happens when hardware like disks or network interfaces stop working
        - Symptoms
            - I/O errors in dmesg
            - Monitoring alerts for NIC or RAID failures
        - Resolution
            - Use SMART tests or check network link lights/switch ports
            - Replace or reseat the faulty component
            - In RAID
                - Mark disk as failed and rebuild array
            - After replacement
                - Verify device status, confirm redundancy, and backup integrity
    - *Data Corruption Issues*
        - Identified by unreadable files, app crashes, or filesystem errors
        - Detection

- fsck run during maintenance
- Logs indicating corrupted files or superblocks

■ Resolution

- Restore from backup or snapshot
- Use fsck with repair options for localized damage
- Investigate causes: bad disk, power loss, or driver issues
- Validate filesystem and application health before resuming operations

○ *Systemd Unit Failures*

■ Service or daemon fails to start or stay active

■ Detection

- systemctl status <service> shows “failed”
- journalctl logs provide failure details

■ Resolution

- Inspect and fix config in /etc/systemd/system/
- Common fixes
  - Correct ExecStart paths or dependencies
- Reload config with systemctl daemon-reload
- Restart service and confirm stability
- Set up monitoring/alerts and document solution

○ *Server Inaccessibility*

■ Occurs when server cannot be accessed via SSH, network, or management interface

■ Detection

- Ping/SSH timeout, out-of-band access fails

■ Resolution

- Determine if network-wide or isolated
- Use console cable or direct access to check system status
- Fix may require reboot, restoring network configs, or repairing service files
- After recovery
    - Validate access via ping, SSH, and management interfaces

- Summary
    - Device failure disrupts access to essential hardware
    - Data corruption destabilizes apps and files, requiring recovery and root cause fixes
    - Systemd unit failures block services and must be debugged and reconfigured
    - Server inaccessibility requires physical access or recovery of network settings
    - Admins resolve issues using system tools, config changes, and recovery procedures to ensure uptime

- **Dependency Issues**
    - Overview
        - Dependencies are required programs, libraries, or files needed for software to function
        - Two common dependency issues
            - Package dependency issues
            - PATH misconfiguration issues
    - *Package Dependency Issues*

- Occur when software fails to install or update due to missing or conflicting libraries
- Recognized by errors like
  - "package X requires Y"
  - "unmet dependencies"
- Check enabled repositories and ensure needed packages are available
- Tools to inspect dependencies
  - yum deplist <package>
  - apt-cache depends <package>
- Fix library version conflicts by updating or downgrading related packages
- Rerun installation and verify application launches without errors

- *PATH Misconfiguration Issues*
  - Occur when the shell cannot locate executables, causing "command not found" errors
  - Detected when
    - which <command>
    - type <command>
      - returns nothing
  - View current search paths
    - echo $PATH
  - Fix by editing
    - /etc/profile
    - /etc/environment
    - ~/.bash_profile
  - Reload shell or log back in after changes
  - Ensure correct directory order in PATH so desired tool is prioritized

- Document PATH changes in configuration management for consistency

- Summary
  - Package dependency issues prevent installations when required components are missing or mismatched
  - PATH misconfigurations break command execution when executables cannot be located
  - Effective troubleshooting involves resolving package sources, aligning versions, and maintaining accurate PATH values

Troubleshooting Networking Issues

Objective 5.3: *Given a scenario, analyze and troubleshoot networking issues on a Linux system*

- **Firewall Issues**
  - Overview
    - Linux servers use firewalls to manage incoming and outgoing network traffic
    - Firewalls ensure only authorized connections reach services
    - Common firewall tools include
      - iptables
      - nftables
      - firewalld
  - *Misconfigured Firewalls*
    - A misconfigured firewall has incorrect rules that allow or block the wrong traffic
    - Common symptoms
      - Services become unreachable
      - Unexpected ports appear open in external scans
    - Causes
      - Typos in rules
      - Incorrect rule order
      - Rules not persisted across reboots
  - *Typo in Rule*
    - Example

- firewall-cmd --add-port=22/tcpp
  - Accepts input but "tcpp" is invalid
  - SSH traffic is blocked

- Detection
  - Use firewall-cmd --list-ports
  - Odd entries like 22/tcpp appear
- Fix
  - Remove with
    - firewall-cmd --remove-port=22/tcpp --permanent
  - Add correct rule
    - firewall-cmd --add-port=22/tcp --permanent
  - Reload
    - firewall-cmd --reload
- Prevention
  - Double-check spelling
  - Use checklists or predefined snippets

- *Incorrect Rule Order*
  - DROP/REJECT rules evaluated before ACCEPT can block legit traffic
  - Example
    - Added HTTP rule
      - iptables -A INPUT -p tcp --dport 80 -j ACCEPT
    - Then set default policy
      - iptables -P INPUT DROP
  - Result
    - HTTP traffic blocked
  - Detection

- iptables -L -n -v shows DROP rule has hits, HTTP rule has none
- Fix
- Insert rule at top
- iptables -I INPUT 1 -p tcp --dport 80 -j ACCEPT

- *Changes Not Persisted*
- Runtime rules vanish after reboot without --permanent flag
- Example
- Added rule
- firewall-cmd --add-service=http
- Works until reboot
- Fix
- Use permanent flag
- firewall-cmd --zone=public --add-service=http --permanent
- Reload
- firewall-cmd --reload

- Summary
- Firewalls control access by allowing/blocking packets based on rules
- Tools
- iptables
- nftables
- firewalld
- Misconfigurations can
- Block valid services
- Allow unwanted access
- Cause rules to disappear after reboot
- Solutions

- Check syntax for typos
- Use correct rule order
- Always apply --permanent and reload

- **Addressing Issues**
  - Overview
    - Addressing problems disrupt network communication in Linux systems
    - Common types of addressing issues
      - DHCP issues
      - IP conflicts
      - Dual stack issues (IPv4 and IPv6 misconfiguration)
  - *DHCP Issues*
    - Symptoms
      - Devices fail to obtain IP addresses
      - Logs show repeated DHCPDISCOVER attempts
      - Interfaces remain unconfigured
    - Detection
      - Check DHCP client status
      - Confirm server is not out of leases
      - Look for "no available leases" errors
    - Fix
      - Ensure DHCP service is running
      - Expand IP address pool if needed
      - Restart DHCP daemon
      - Force clients to request a new lease

- Post-fix
  - Confirm clients obtain IP addresses
  - Update network documentation

- *IP Conflicts*
  - Symptoms
    - Duplicate address warnings in syslog
    - ARP conflict messages
    - Random disconnects and degraded performance
  - Detection
    - Check DHCP lease files
    - Cross-reference with DNS and static IP assignments
  - Fix
    - Assign unique IP to one device
    - Update static configurations
    - Clear ARP cache
  - Post-fix
    - Monitor network to ensure conflict is resolved
- *Dual Stack Issues*
  - Symptoms
    - IPv6 works but IPv4 fails, or vice versa
    - Ping tests only succeed on one protocol
    - Application logs show
      - address already in use
      - bind
        - Cannot assign requested address
  - Detection

- Check if /etc/sysctl.conf disables IPv6
- Verify DNS has both A (IPv4) and AAAA (IPv6) records

■ Fix

- Re-enable IPv6 if required
- Update service config to bind to both IPv4 and IPv6:
  - 0.0.0.0 and ::
- Ensure firewalls allow traffic for both protocols

■ Post-fix

- Test dual-protocol connectivity

○ Summary

■ DHCP Issues

- Caused by lease failures or server misconfig
- Fix by restarting DHCP, expanding pools, and requesting new leases

■ IP Conflicts

- Caused by duplicate IP assignments
- Fix by assigning unique addresses and clearing ARP caches

■ Dual Stack Issues

- IPv4/IPv6 misconfiguration
- Fix by checking sysctl, DNS, service bindings, and firewall rules

- **Routing Issues**

○ Overview

■ Routing determines how network packets move between systems

- Misconfigurations can make servers unreachable or isolate them from services
- Three common routing issues
  - DNS issues
  - Routing issues with the gateway
  - Server unreachable

- *DNS Issues*
  - Hostnames fail to resolve into IP addresses
  - Example symptom
    - ping server.example.com returns “unknown host”
  - Diagnostic tools
    - Check /etc/resolv.conf
    - Use ping or DNS lookup utilities
  - Common causes
    - Misconfigured or unreachable DNS server
    - Outdated search domain
    - Typo in hostname
  - Fix
    - Correct DNS IP in /etc/resolv.conf
    - Verify DNS service is reachable
    - Retest resolution with ping or dig
- *Routing Issues with the Gateway*
  - Packets fail to leave the local network
  - Servers can talk to local peers but not external resources
  - Diagnose with
    - ip route

- route -n
- Look for missing or incorrect default route 0.0.0.0/0
- Often caused by
  - Network changes
  - DHCP misconfiguration
- Fix
  - Manually add or update default route
  - Make changes persistent in network config files
  - Verify with ping to an external IP

- *Server Unreachable*
  - Neither hostname nor IP responds to pings or SSH
  - Steps to diagnose
    - ip link to check interface status
    - ip addr to confirm assigned IP
  - Check for
    - Incorrect switch port or VLAN settings
    - Firewall rules blocking ICMP or SSH
  - Fix
    - Restart network service
    - Correct physical or VLAN misconfigurations
    - Adjust firewall to allow essential traffic
  - Confirm with successful ping or SSH login
- Summary
  - DNS issues break name resolution and are resolved by fixing /etc/resolv.conf

- Gateway issues prevent outbound traffic and require default route updates
- Server unreachable errors stem from network interface, switch/VLAN, or firewall misconfigurations
- Diagnosing and fixing these issues ensures stable and reliable network connectivity

- **Interface Misconfiguration**
  - Overview
    - Interface misconfiguration disrupts server connectivity and performance
    - Five common interface misconfiguration issues
      - Subnet misconfiguration
      - MTU mismatch
      - Cannot ping server
      - Interface bonding issues
      - MAC spoofing problems
  - *Subnet Misconfiguration*
    - Occurs when interface IP or netmask does not match the intended subnet
    - Detection
      - Inability to communicate with hosts on the same network
      - ip addr show reveals mismatched subnet (e.g., 192.168.1.100/24 vs 192.168.2.0/24)
    - Fix
      - Edit interface configuration to correct IP/netmask
      - Apply changes with netplan apply or systemctl restart networking

        - Test connectivity with ping
    - *MTU Mismatch*
        - Happens when endpoints use different maximum transmission units
        - Symptoms
            - Large transfers hang or time out
            - ping -s 1500 shows “Frag needed but DF set” errors
        - Detection
            - Compare MTUs with ip link show
        - Fix
            - Set matching MTUs on both endpoints (e.g., 1500)
            - Update interface configurations
            - Retest with ping or file transfer
    - *Cannot Ping Server*
        - Indicates issues like disabled interface, missing IP, or ICMP blocked
        - Symptoms
            - ping <server_ip> returns “Destination Host Unreachable” or times out
        - Fix
            - Check if interface is up with ip link
            - Bring up with ip link set <interface> up
            - Assign correct IP if needed
            - Verify ICMP isn’t blocked using ufw status or iptables -L
            - Confirm with ping after fixing
    - *Interface Bonding Issues*
        - Result from improper NIC teaming configurations
        - Symptoms

- One interface under bond0 marked down
- Combined throughput not achieved

■ Modes

- Mode 0: balance-rr
- Mode 1: active-backup
- Mode 4: 802.3ad (LACP)

■ Fix

- Confirm bonding driver is loaded
- Check bond config in /etc/netplan/*.yaml or /etc/sysconfig/network-scripts/ifcfg-bond0
- Match bond mode with switch setup
- Restart networking
- Verify both NICs are up under the bond
- Test failover by unplugging one NIC

○ *MAC Spoofing Problems*

■ Caused by duplicate or unauthorized MAC addresses

■ Symptoms

- arping <server_ip> returns multiple MACs
- ip neigh shows MAC flapping

■ Detection

- Inspect MACs with ip link show

■ Fix

- Ensure unique MAC for each interface or bond
- Restart networking
- Confirm stability with ip neigh show

○ Summary

- Subnet Misconfiguration
  - Mismatched IP/netmask blocks communication
- MTU Mismatch
  - Inconsistent packet sizes cause drops or fragmentation
- Cannot Ping Server
  - Interface down, misconfigured, or ICMP blocked
- Interface Bonding Issues
  - Misaligned bond settings or unsupported modes
- MAC Spoofing Problems
  - Duplicate MACs cause flapping and instability

- **Link Issues**
  - Overview
    - Link issues disrupt network communication between devices
    - Two common types of link issues
      - Link down
      - Link negotiation issues
  - *Link Down*
    - Occurs when a network interface fails to establish or maintain a connection
    - Detected using ip a or ifconfig showing status as “DOWN”
    - Logs show “Link is Down” or “eth0: Link is Down” via dmesg or journalctl
    - Causes include
      - Loose cables
      - Faulty ports

- Driver issues
- Misconfiguration

■ Fix

- Ensure cable is connected and switch/router ports are functional
- Run ip link show <interface> to check admin status
- Bring up interface using ip link set <interface> up
- Reload driver with modprobe if needed
- Check for firmware updates
- Restart network service using systemctl restart network
- Check for kernel module conflicts

○ *Link Negotiation Issues*

■ Occur when devices cannot agree on connection speed or duplex mode

■ Symptoms

- Poor performance
- Slow speeds
- Connectivity dropouts

■ Detected using ethtool <interface> to view speed, duplex, and autonegotiation status

■ Causes include

- One device with manual settings, another on auto
- Mismatched speed or duplex modes
- Faulty cables or NICs
- Driver bugs

■ Fix

- Match speed and duplex settings on both devices
- Prefer enabling autonegotiation on both ends

- Replace faulty cables or network cards
- Review logs for hardware/driver errors
- Update or change network drivers if needed

- Summary
  - Link Down
    - Interface disconnected due to physical, driver, or configuration issues
  - Link Negotiation
    - Devices fail to align on speed/duplex, degrading performance
  - Use diagnostic tools
    - ip a
    - ip link show
    - ethtool
    - journalctl, dmesg
  - Resolve with interface configuration checks, hardware tests, and driver updates

Troubleshooting Security Issues

Objective 5.4: *Given a scenario, analyze and troubleshoot security issues on a Linux system*

- **SELinux Issues**
  - Overview
    - SELinux enforces access control policies to restrict user and application actions
    - Provides strong protection against unauthorized access and malicious activity
    - Three core components of SELinux
      - Policy
      - Context
      - Booleans
  - *SELinux Policy Issues*
    - Define rules for what actions are allowed or denied
    - Misconfigured or overly restrictive policies can block legitimate actions
    - Log message clue
      - avc: denied entries
    - Troubleshooting tools
      - ausearch
      - sealert
    - Fixes include modifying policy rules or creating custom policy modules
    - Policy changes should be tested in controlled environments
  - *SELinux Context Issues*

- Every file, process, and resource is labeled with a context
- Incorrect labels prevent access or cause application failures
- Check file context
  - ls -Z /etc/httpd/conf/httpd.conf
- Example issue
  - file mislabeled user_home_t instead of httpd_config_t
- Fix with
  - sudo restorecon -v /etc/httpd/conf/httpd.conf
- Periodic use of restorecon on key directories prevents mislabeling

- *SELinux Boolean Issues*
  - Booleans are toggles for adjusting security behavior without editing policy
  - Allow temporary or situational changes to security settings
  - Check all booleans
    - getsebool -a
  - Modify boolean
    - setsebool -P httpd_can_sendmail 1
  - Use -P to make changes persistent
  - Useful for adjusting permissions like allowing Apache to send mail
  - Excessive boolean changes can weaken security and should be used cautiously
- Summary
  - SELinux protects systems by enforcing detailed access control policies
  - Policy issues cause access denials and require log analysis and rule adjustments

        - Context issues involve incorrect labeling of files or processes and are fixed with restorecon
        - Boolean toggles allow flexible adjustments without rewriting policies
        - Understanding and managing policies, contexts, and booleans helps maintain security and functionality in Linux environments

- **File and Directory Permission Issues**
    - Overview
        - Linux uses file and directory permissions to control user access
        - Permissions regulate read, write, and execute capabilities
        - Two major permission mechanisms
            - Attributes
            - Access Control Lists (ACLs)
    - *Attributes*
        - File attributes apply behavior restrictions beyond standard permissions
        - Misconfigured attributes may block modification or deletion
        - Example of restrictive attribute
            - immutable
        - Check attributes using
            - lsattr
        - Common attributes
            - i for immutable
            - a for append-only
        - Remove attributes using
            - chattr -i /path/to/file

- -i removes the immutable flag
- Removing the immutable attribute allows the file to be modified or deleted

- *Access Control Lists (ACLs)*
  - ACLs offer fine-grained permission control per user or group
  - Allow more detailed access than owner-group-other structure
  - Use getfacl to view ACL entries
  - Use setfacl to modify ACL permissions
  - Example command
    - setfacl -m u:john:r /path/to/file
      - -m modifies the ACL entry
      - u:john:r grants read-only access to user john
  - ACLs do not replace base permissions but extend them
  - ACL misconfigurations can result in access denial or privilege escalation
  - Regular ACL audits ensure proper permission settings
- Summary
  - Attributes control file behaviors like immutability
  - Use lsattr to view and chattr to modify attributes
  - ACLs provide detailed access settings beyond standard modes
  - Use getfacl and setfacl to inspect and modify ACL entries
  - Proper management of both attributes and ACLs maintains system security and usability

- **Access Issues**
    - Access Issues
        - Linux systems manage access like secured buildings
        - Three primary access issue types
            - Account access issues
            - Remote access issues
            - Certificate issues
    - *Account Access Issues*
        - Occur when users cannot log in
        - Common causes
            - Incorrect credentials
            - Locked or disabled accounts
        - Check system logs for
            - "Authentication failure"
            - "Account locked"
        - Commands to resolve issues
            - Reset password
            - sudo passwd john
        - Check account status
            - sudo passwd -S john
        - Unlock account
            - sudo passwd -u john
        - Re-enable disabled account:
            - sudo usermod -e '' john
        - Regular monitoring ensures smooth login access

- *Remote Access Issues*
  - Arise when users fail to connect via SSH or VPN
  - Possible causes
    - SSH service not running
    - Firewall blocking ports
    - Misconfigured SSH settings or missing keys
  - Troubleshooting steps
    - Check SSH status
      - sudo systemctl status sshd
    - Check firewall rules
      - sudo ufw status
      - sudo iptables -L
  - Confirm port 22 is open for SSH
  - Check routing and SSH key presence
- *Certificate Issues*
  - Occur when SSL/TLS certificates are invalid or misconfigured
  - Symptoms
    - "SSL certificate expired"
    - "SSL handshake failure"
  - Causes
    - Expired certificates
    - Improper certificate chains
    - Issues with Certificate Authority (CA)
  - Diagnostic command
    - openssl s_client -connect yourserver.com:443
  - Fixes

        - Renew expired certificates via provider
        - Create self-signed certificates using OpenSSL
        - Ensure full certificate chain and intermediates are installed
    - Regular checks prevent communication breakdowns
- Summary
    - Account issues are resolved using passwd, usermod, and log analysis
    - Remote access issues require checking SSH service, firewall rules, and port access
    - Certificate issues are fixed by renewing or properly configuring certificates with tools like OpenSSL
    - Maintaining secure access depends on monitoring user credentials, remote services, and certificate validity

- **Configuration Issues**
    - Overview
        - Proper configuration is essential for Linux system security and functionality
        - Two common configuration issues
            - Exposed or misconfigured services
            - Misconfigured package repositories
    - *Exposed or Misconfigured Services*
        - Occur when system services are accessible from the internet without proper restrictions
        - Example
            - Public SSH access without firewall rules

- Risks
  - Unauthorized access
  - Brute-force attacks
  - Security vulnerabilities
- Detection methods
  - Review system logs
  - Use port scanning tools like nmap
- Resolution steps
  - Restrict access via firewall (e.g., only allow trusted IPs)
  - Disable unused services
  - Limit accessibility of critical services to internal use only

- *Misconfigured Package Repositories*
  - Occur when repository URLs are incorrect, outdated, or unreachable
  - Causes failed software updates or installations
  - Symptoms
    - sudo apt update or yum update returns repository errors
  - Detection
    - Read error messages related to unreachable repositories
  - Resolution steps
    - Check repository configuration files:
      - Debian-based
        - /etc/apt/sources.list
      - RHEL-based
        - /etc/yum.repos.d/
    - Correct or replace invalid URLs with working mirrors
    - Verify authentication keys are properly set up

        - Regular audits ensure reliable software update access
    - Summary
        - Exposed services create external security risks and must be firewall-restricted or disabled
        - Misconfigured repositories disrupt updates and must be corrected in system configuration files
        - Continuous monitoring and validation of service settings and repo URLs preserve system stability and security

- **Vulnerabilities**
    - Overview
        - Vulnerabilities are weaknesses or flaws in the system that attackers can exploit
        - Three common vulnerability types
            - Unpatched vulnerable systems
            - Use of obsolete or insecure protocols and ciphers
            - Cipher negotiation issues
    - *Unpatched Vulnerable Systems*
        - Occur when the system has not been updated with the latest security patches
        - Risks
            - Exposure to known exploits and security flaws
        - Detection methods
            - Review outdated packages
            - Use vulnerability scanners

- Resolution
  - Debian-based systems: sudo apt update && sudo apt upgrade
  - Red Hat-based systems: sudo yum update
- Regular updates ensure protection against known threats

- *Obsolete or Insecure Protocols and Ciphers*
  - Older encryption algorithms or protocols pose cryptographic risks
  - Examples
    - SSLv3 vulnerable to POODLE attacks
    - RC4 cipher vulnerable to RC4 bias attacks
  - Detection
    - Review server configuration files such as sshd_config or apache2.conf
  - Resolution
    - Disable SSLv3 and RC4
    - Use strong ciphers (e.g., AES)
    - Use secure protocols (e.g., TLS 1.2 or 1.3)
- *Cipher Negotiation Issues*
  - Occur when systems fail to agree on strong cipher suites
  - Risks
    - Downgrade attacks forcing fallback to weak encryption
  - Cipher suite = key exchange + encryption algorithm + MAC
  - Example of weak suite
    - TLS_RSA_WITH_RC4_128_MD5
      - RC4 is insecure
      - MD5 is vulnerable to collision attacks
  - Resolution

- Review logs for weak cipher use
- Configure systems to reject outdated SSL/TLS versions
- Enforce use of strong cipher suites on both client and server
- Regularly audit and update cipher configurations

- Summary
  - Unpatched systems must be updated to avoid known exploits
  - Obsolete protocols and weak ciphers must be disabled to prevent cryptographic attacks
  - Cipher negotiation must enforce secure suites to prevent downgrade exploitation
  - Regular updates, configuration audits, and encryption enforcement protect system integrity

Troubleshooting Performance Issues

Objective 5.5: *Given a scenario, analyze and troubleshoot performance issues*

- **CPU Issues**
  - Overview
    - CPU issues can cause performance degradation or system unresponsiveness
    - Four types of CPU-related problems
      - High CPU usage
      - High load average
      - High context switching
      - CPU bottleneck
  - *High CPU Usage*
    - Occurs when the processor is heavily used by running processes
    - Indicators
      - top or htop shows usage over 80-90% for extended periods
    - Causes
      - Resource-heavy processes (e.g., java, python)
      - Unnecessary background services
    - Resolution
      - Identify high-usage processes
      - Optimize or terminate them
      - Disable unnecessary services
      - Example command

- systemctl stop <service>

- *High Load Average*
  - Happens when too many processes are waiting to execute
  - Indicators
    - uptime or top shows a load average exceeding number of CPU cores
  - Causes
    - Excessive concurrent processes
    - Specific tasks overloading the CPU
  - Resolution
    - Investigate bottleneck processes (e.g., slow queries)
    - Offload tasks or scale system (add CPUs)
- *High Context Switching*
  - Occurs when the CPU constantly switches between processes
  - Context
    - Process state (CPU registers, memory, etc.)
    - Context states include running, waiting, and stopped
  - Indicators
    - Use vmstat or pidstat
    - 500-1000+ switches per second considered high
  - Causes
    - Too many threads or processes
  - Resolution
    - Reduce thread count or number of concurrent processes
    - Optimize application/thread management
    - Limit web server child processes if applicable

- *CPU Bottleneck*
  - Occurs when CPU becomes the limiting factor in system performance
  - Indicators
    - CPU usage remains over 80-90%
    - Load average exceeds number of cores
    - Tasks take more than 5-10 seconds to complete
  - Tools
    - top or htop to identify heavy CPU consumers
  - Resolution
    - Optimize applications
    - Improve code efficiency
    - Add more CPU cores or upgrade server hardware
    - Distribute workload across multiple systems
- Summary
  - High CPU usage
    - Processor overwhelmed by resource-hungry tasks
  - High load average
    - Too many queued tasks exceeding CPU capacity
  - High context switching
    - Inefficient CPU time spent switching between tasks
  - CPU bottleneck
    - CPU limits overall performance due to overload
  - Solutions include optimization, process management, and hardware upgrades to ensure smooth system performance

- **Memory Issues**
    - Overview
        - Memory issues can degrade performance or cause application crashes
        - Two main memory-related issues
            - Swapping
            - Out of Memory (OOM) errors
    - *Swapping*
        - Occurs when system runs out of RAM and starts using disk as virtual memory
        - Swap space located in /swapfile or swap partition
        - Use swapon -s to view active swap
        - Indicators
            - Noticeable system slowdown
            - top or htop shows sustained swap usage over 10–20%
        - Cause
            - System moves inactive data to disk to free up RAM for active processes
        - Monitoring tools
            - free -h
            - vmstat
        - Resolution
            - Add more RAM if needed
            - Adjust swappiness to reduce swap usage
            - Example
                - sudo sysctl vm.swappiness=10 to prefer RAM over swap
    - *Out of Memory (OOM) Errors*

- Occurs when both RAM and swap are exhausted
- OOM Killer terminates processes to prevent crash
- Indicators
  - Unexpected termination of processes
  - OOM logs in system logs
- Monitoring tools
  - free -h
  - top
- Resolution
  - Optimize application memory usage
  - Increase RAM or swap space
  - Adjust overcommit settings
- Overcommit
  - Kernel feature allowing memory allocation beyond physical limits

- Summary
  - Swapping
    - Performance slowdown when disk is used for memory
  - OOM
    - Critical process termination due to complete memory exhaustion
  - Mitigation
    - Monitor memory regularly
    - Use swappiness and overcommit settings wisely
    - Scale physical memory or swap space as needed

- **Disk I/O Issues**
    - *Disk I/O Issues*
        - Disk I/O issues occur when delays in reading from or writing to storage devices affect system performance
        - Comparable to workers waiting at a slow filing cabinet in a busy office
        - Three common problems
            - High input/output wait time
            - High disk latency
            - Slow remote storage response
    - *High Input/Output Wait Time*
        - Occurs when processes wait too long for data from the disk
        - Measured using commands like top or iostat
        - High I/O wait is indicated by sustained values of 10%–20% or more
        - Causes
            - Overloaded disks
            - Slow storage
        - Tools
            - iotop
            - dstat
            - topiostat
        - Solutions
            - Optimize disk-heavy processes
            - Spread out I/O load
            - Upgrade to faster storage (e.g., SSDs)
    - *High Disk Latency*
        - Refers to delay in the disk's response to read/write requests

- Causes
  - Disk hardware issues
  - Too many concurrent requests
  - Inefficient configurations
- Detected using
  - iostat
  - sar
- Normal latency
  - Under 10ms
- Problematic latency
  - Over 20–30ms
- Solutions
  - Check disk errors
  - Update drivers
  - Improve RAID or disk configurations
  - Upgrade disk hardware or file system usage

- *Slow Remote Storage Response*
  - Delays in accessing networked storage (e.g., NFS, SAN, cloud)
  - Causes
    - Network bottlenecks
    - Misconfigured storage systems
  - Symptoms
    - Slow file access from remote storage
    - Long response times
  - Tools
    - nfsstat

        - netstat
        - ping
        - netperf
    - Solutions
        - Improve network infrastructure (e.g., upgrade to 10GbE)
        - Optimize storage system settings
        - Use more efficient storage protocols
- Summary
    - Disk I/O issues impact performance when storage device access is delayed
    - High I/O wait
        - Processes blocked waiting for data
    - High disk latency
        - Slow response to read/write due to hardware or config issues
    - Slow remote storage response
        - Network/storage inefficiencies in external systems
    - Identifying and resolving these issues leads to faster, more reliable system operations

- **Network Stability Issues**
    - Overview
        - Stable network communication is essential for application and service operation
        - Three common network stability issues
            - Packet drops

- Random disconnects
- Random timeouts

- *Packet Drops*
  - Occur when data packets fail to reach their destination
  - Identified using tools like ping or netstat
  - Example
    - ping -c 100 <destination>
      - Shows response rate below 100%
  - Problematic when packet loss exceeds 1–2%
  - Causes
    - Network congestion
    - Faulty cables
    - Hardware issues (e.g., switches/routers)
  - Resolution
    - Check hardware
    - Use ifconfig or ethtool to inspect interface errors
    - Adjust Quality of Service (QoS) settings or increase bandwidth
- *Random Disconnects*
  - Happen when a network connection is unexpectedly terminated
  - Indicated by
    - Logs showing "connection reset" or "connection closed"
    - Users/services suddenly losing access
  - Causes
    - Faulty cables or network hardware
    - Misconfigured network stack
    - Firewalls/security tools closing connections

        - Resolution
            - Inspect logs with dmesg, ifconfig
            - Verify firewall settings
            - Adjust TCP settings (e.g., MSS)
            - Update drivers and firmware
    - *Random Timeouts*
        - Occur when the system fails to receive a response in time
        - Indicated by timeout errors in logs or output
        - Example
            - curl <url> returns “connection timed out”
        - Causes
            - Network congestion
            - DNS misconfiguration
            - Overloaded server
        - Acceptable timeout threshold: 5–10 seconds
        - Resolution
            - Use ping or traceroute to identify bottlenecks
            - Verify DNS configuration
            - Check server performance
            - Adjust TCP timeout values or upgrade infrastructure
    - Summary
        - Packet drops
            - Lost packets from congestion or hardware faults
        - Random disconnects
            - Unexpected termination of network connections
        - Random timeouts

        - Delays preventing timely response from services
    - Mitigation
        - Monitor tools like ping, traceroute, dmesg
        - Inspect hardware and network stack
        - Adjust configuration and upgrade resources when needed

- **Network Performance Issues**
    - Overview
        - Network performance affects the speed and reliability of communication
        - Four common performance issues
            - High latency
            - Jitter
            - Slow response times
            - Low throughput
    - *High Latency*
        - Delay in data travel between two points on a network
        - Detected using ping or traceroute
        - Thresholds
            - Over 100ms for local = high
            - Over 300ms for remote = high
        - Causes
            - Network congestion
            - Faulty hardware (e.g., cables, switches)
            - Misconfigured routing
        - Resolution

        - Optimize network path
        - Upgrade infrastructure (e.g., from 1Gbps to 10Gbps)
- *Jitter*
    - Variation in latency over time
    - Affects real-time apps like voice or video
    - Acceptable jitter
        - Under 30ms
    - Detection
        - Use ping -i 0.2 <destination> to observe variations
    - Causes
        - Network congestion
        - Hardware instability
    - Resolution
        - Check routers/switches
        - Implement QoS policies
        - Use stable network paths
- *Slow Response Times*
    - Network service takes too long to respond
    - Indicators
        - Web
            - >1–2 seconds
        - Database
            - >5–10 seconds
    - Causes
        - Latency or congestion
        - Overloaded server

- Poor application performance
- Detection
  - Use
    - curl
    - wget
- Resolution
  - Optimize code
  - Reduce server load
  - Check CPU, RAM, and configs

- *Low Throughput*
  - Insufficient data transfer rate
  - Expected performance
    - ≥80% of bandwidth
  - Example
    - 1Gbps link only gets 100–200Mbps
  - Tools
    - Use iperf to measure throughput
  - Causes
    - Bandwidth limits
    - Faulty cables
    - Misconfigurations
  - Resolution
    - Check and upgrade NICs, switches
    - Remove unnecessary traffic
    - Optimize network routes
- Summary

- High latency
  - Long delays
- Jitter
  - Inconsistent delays
- Slow response
  - Services too slow to respond
- Low throughput
  - Poor data transfer rate
- Tools
  - ping, traceroute, iperf, curl, wget
- Fixes
  - Optimize routes, update hardware, configure QoS, balance load

- **System Responsiveness Issues**
  - Overview
    - Good responsiveness maintains system performance and productivity
    - Four common system responsiveness issues
      - Slow application response
      - Sluggish terminal behavior
      - Slow startup
      - System unresponsiveness
  - *Slow Application Response*
    - Application takes too long to process input or return results
    - Recognized when apps (e.g., web server, database) delay >1–2 seconds
    - Detection tools

- top
- htop

- Causes
  - High CPU or memory usage
  - I/O bottlenecks
  - Competing background processes
- Resolution
  - Optimize application code
  - Add RAM or CPU
  - Stop unnecessary services

- *Sluggish Terminal Behavior*
  - Commands in the terminal respond slowly
  - Recognized when basic commands (e.g., ls) take several seconds
  - Causes
    - CPU
    - Memory
    - Disk I/O bottlenecks
  - Detection tools
    - top
    - iotop
  - Resolution
    - Optimize running processes
    - Clean up system resources
    - Add memory or CPU power
    - Limit background workloads
- *Slow Startup*

- System or apps take too long to boot or become ready
- Recognized when
  - System boots >2–3 minutes
  - Apps take >5 minutes to load
- Detection tools
  - systemd-analyze
- Causes
  - Too many services
  - Misconfigured startup settings
- Resolution
  - Use systemctl to disable or delay unnecessary services
  - Optimize configuration files
  - Reduce startup programs

- *System Unresponsiveness*
  - System freezes and stops accepting input
  - Recognized by lack of user interaction for 5–10+ minutes
  - Detection tools
    - dmesg
    - journalctl
    - top
    - htop
  - Causes
    - Resource exhaustion (RAM or CPU)
    - Kernel panic or runaway processes
  - Resolution
    - Monitor system usage

        - Kill runaway processes
        - Add physical RAM or upgrade CPU
        - Ensure balanced resource allocation
    - Summary
        - Slow application response
            - Delayed user interactions
        - Sluggish terminal
            - Laggy command execution
        - Slow startup
            - Long boot or app load times
        - System unresponsiveness
            - Full system freeze
        - Tools
            - top, htop, iotop, systemd-analyze, dmesg, journalctl
        - Fixes
            - Optimize resources, trim background tasks, upgrade hardware

- **Process Management Issues**
    - Overview
        - Efficient process management ensures performance and stability
        - Two common issues
            - Blocked processes
            - Exceeding baselines
    - *Blocked Processes*
        - Occur when a process waits for a resource or system lock

- Indicated by commands or applications getting stuck
- Processes in D (uninterruptible sleep) state signal blocking
- Causes
    - Waiting on disk I/O, memory, or another process
    - Resource contention
    - System-level locks
- Detection tools
    - ps
    - top
    - lsof
    - strace
- Resolution
    - Optimize disk I/O
    - Add memory
    - Resolve inter-process dependencies
    - Tune application or database lock settings

- *Exceeding Baselines*
    - Process uses more resources than normal or expected
    - Can cause overall performance degradation
    - Signs include
        - CPU or memory use >80% for extended time
        - High disk activity from a single process
    - Detection tools
        - top
        - htop
        - pidstat

        - Resolution
            - Identify inefficient code or memory leaks
            - Reduce number of processes
            - Use ulimit to set resource caps
            - Add RAM or CPU as needed
    - Summary
        - Blocked processes
            - Stuck waiting on resources or locks
        - Exceeding baselines
            - Excessive use of CPU, memory, or I/O
        - Use tools
            - ps, top, htop, lsof, strace, pidstat
        - Fixes
            - Optimize usage, resolve contention, enforce limits, upgrade resources

- **Security-related Performance Issues**
    - Overview
        - Security threats can directly impact system performance and stability
        - Two common issues
            - High failed log-in attempts
            - Hardware errors
    - *High Failed Log-in Attempts*
        - Caused by brute force or unauthorized access attempts
        - Can increase system load or lead to account lockouts

- Recognition methods
  - Review
    - /var/log/auth.log
    - journalctl
  - Monitor for >5–10 failed attempts from the same IP in 10–15 minutes
- Countermeasures
  - Use fail2ban to block abusive IPs
  - Enforce strong password policies
  - Enable multi-factor authentication (MFA)
  - Limit access with firewalls or IP allowlisting

- *Hardware Errors*
  - Include failing disks, network cards, or other physical components
  - Lead to I/O errors, unresponsiveness, or degraded performance
  - Detection tools
    - dmesg or journalctl to find system errors
    - smartctl for disk health
    - ethtool for network diagnostics
  - Resolution
    - Replace failing hardware
    - Migrate data from problematic devices
    - Set up predictive failure alerts and monitoring
- Summary
  - High failed log-in attempts suggest possible attacks that can exhaust resources
  - Hardware failures affect stability and must be proactively diagnosed

- Use tools like fail2ban, smartctl, ethtool, journalctl
- Regular monitoring improves system resilience and prevents downtime

Conclusion

- **Conclusion**
    - Course Summary
        - Course covered all 5 domains of the CompTIA Linux+ (XK0-006) exam
        - Objectives were taught in a reorganized format to aid understanding
        - All objectives and subtopics were included despite not being in sequential order
        - Each video title is labeled with relevant objective numbers for easy reference
    - Domain Review
        - Domain 1: System Management (23%)
            - Basic Linux concepts
            - Device and storage management
            - Network services
            - Shell operations
            - Virtualization
            - Backup and restore
            - Topics include boot process, kernel modules, partitioning, volumes, shell commands, VMs, archiving
        - Domain 2: Services and User Management (20%)
            - File and directory management
            - Local account management
            - Process and job management
            - Software management

- Containers
- systemd
- Topics include user management, package control, process control, containerization

- Domain 3: Security (18%)
  - Authentication, authorization, accounting
  - Firewall implementation
  - OS and account hardening
  - Cryptography
  - Compliance and audit procedures
  - Topics include logging, NAT, privilege management, access control, encryption, vulnerability detection
- Domain 4: Automation, Orchestration, and Scripting (17%)
  - Automating tasks
  - Python language
  - Git repositories
  - Artificial Intelligence
  - Topics include IaC, orchestration tools, shell scripting, Git usage
- Domain 5: Troubleshooting (22%)
  - System configuration
  - Troubleshooting methods
  - Topics include data acquisition, hardware, storage, networking, performance issues

- Exam Voucher Tips
  - Take exam at PearsonVue test center or online with OnVue
  - Exam voucher must be purchased before scheduling

- Options for voucher purchase
  - Directly through PearsonVue or CompTIA store (full price)
  - Discounted vouchers from diontraining.com/vouchers
    - Save 10%
    - Includes free access to searchable CompTIA video library
    - Available in over 50 countries

- Top 5 Tips for the Exam
  - 1. Use a Cheat Sheet
    - Ask for a dry erase board (test center) or use the digital whiteboard (online)
    - Use first few minutes to write down critical info (commands, switches, etc.)
  - 2. Skip the Sims
    - Performance-Based Questions (PBQs) come first and are time consuming
    - Mark them for review, complete multiple-choice questions first
    - Return to PBQs later with more confidence
  - 3. Take a Guess
    - No penalty for wrong answers
    - Eliminate incorrect options and guess from remaining choices
  - 4. Pick the Best Time
    - Choose time based on personal energy level (e.g., 10am)
    - Avoid scheduling after a long workday
    - Arrive early (20–30 mins), be prepared and relaxed
    - Note: Online proctoring does not allow restroom breaks once the exam starts

        - 5. Be Confident
            - Study videos, quizzes, study notes before the exam
            - If unsure, take practice exams to build confidence
            - Use the included full-length practice exam
            - Additional 6 practice exams available in separate course
    - Final Recommendations
        - Don’t memorize answer keys
        - Understand the reasoning behind correct and incorrect answers
        - Use provided explanations to improve understanding
        - Schedule the exam once confident with practice exam results
        - Share certification success in Q&A, Facebook group, or tag on LinkedIn

- **BONUS: Where to go from here?**
    - Completion Acknowledgment
        - Completion of the course demonstrates dedication and commitment to professional growth
        - Course experience intended to be both valuable and enjoyable
        - Continued success anticipated in the IT and cybersecurity industry
    - Exclusive Student Offers
        - Special offers prepared for continuing education
        - Designed to help expand skills beyond Linux+
    - Course Options from Dion Training
        - Courses available on both Udemy and diontraining.com
        - Certification courses include
            - CompTIA Security+

- CompTIA CySA+
- CompTIA PenTest+
- CompTIA SecurityX
- ITIL4
- PRINCE2

- New content added regularly to remain current with industry trends

- Continuing Education
  - Advanced training in cybersecurity, IT service management, and project management
  - Ideal for staying competitive and knowledgeable in a changing field
- Udemy Pricing and Discounts
  - Udemy course pricing varies by region and demand
  - diontraining.com/udemy provides updated links for lowest prices
- Website Course Features (Diontraining.com)
  - Complete study packages per certification
  - Includes
    - Video training
    - Quizzes
    - Study guides
    - Full-length practice exams
    - Exam vouchers (for many certifications)
  - Offers a 100% pass guarantee
    - Retake exam cost covered if course is completed and exam not passed on first attempt
- Hands-On Labs

- Website courses for CompTIA certifications include 40 hours of integrated labs
- Labs provide guided, hands-on experience for real-world preparation

- Exam Vouchers
  - Available for purchase through diontraining.com
  - Voucher tailored to selected certification and exam location
  - Option to purchase "take two" vouchers for discounted retake opportunity
- Support and Community
  - Support available for certification preparation and career growth
  - Connect through
    - facebook.com/groups/diontraining
    - diontraining.com