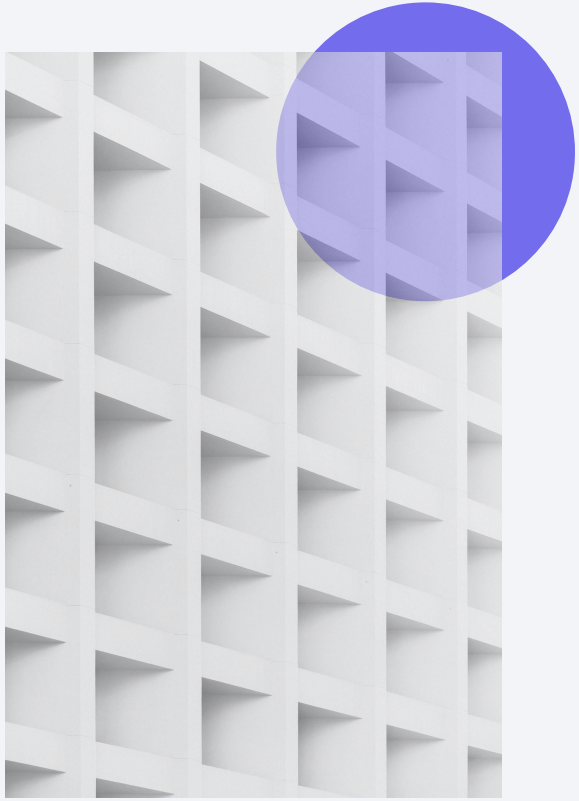
September 27, 2025

Introduction to UNIX OS & Commands



Agenda

Intro to UNIX OS & Commands

UNIX File System Overview

Pipe Concept: Std Input/Output/Error

Unix Environment & Shells

Filters: Simple & Advanced

Basic OS Commands for Monitoring

Vi Editor Basics

Shell Programming Fundamentals

Use Cases: File Ops, SFTP & DB

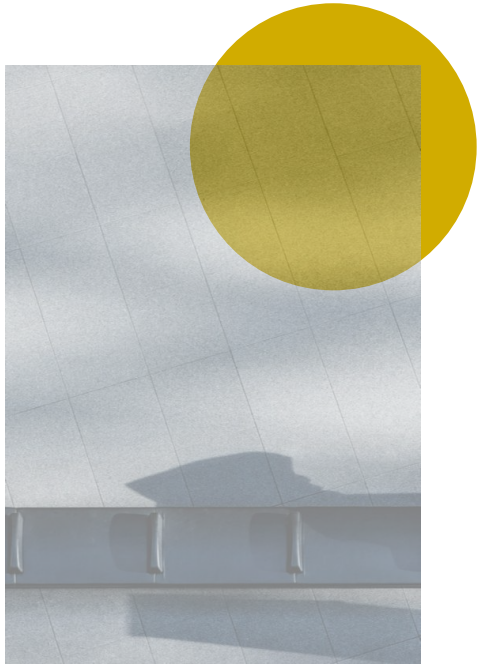
Introduction

Introduction to UNIX Operating System

- UNIX was developed in the late 1960s at AT&T Bell Labs by Ken Thompson, Dennis Ritchie, and others.
- It introduced a simple, modular design with a hierarchical file system and powerful shell environment.
- UNIX supports multiple users and multitasking, making it ideal for servers and workstations.
- Its design principles influenced many modern operating systems including Linux, BSD, and macOS.
- Key features include portability across hardware, a rich set of command-line tools, and robust security model.

Architecture

Operating System Architecture & Components



Kernel

The kernel is the core of UNIX, managing hardware resources, memory, processes, and system calls. It operates in privileged mode and controls all low-level tasks.

Shell

The shell acts as an interface between the user and the kernel. It interprets user commands, executes programs, and provides scripting capabilities for automation.

User Programs & File System

User programs run in user space and access system resources via the shell and kernel. The file system manages data storage, organizing files and directories with hierarchical structure.

Commands

Basic UNIX Commands

- ls: Lists files and directories in the current directory. Use 'ls -l' for detailed view.
- cd: Changes the current directory. Example: 'cd /home/user' moves to the user directory.
- pwd: Prints the working directory, showing the full path of the current directory.
- mkdir: Creates a new directory. Example: 'mkdir new_folder' creates a folder named new_folder.
- rm: Removes files or directories. Use 'rm -r' to remove directories recursively.

Creating a Shell Script: Hello World

Steps to Create and Run a Shell Script

- Write the shell script in a text editor, starting with the shebang line '#!/bin/bash' to specify the shell.
- Add the command to print 'Hello World' using: echo "Hello World"
- Save the file with a '.sh' extension, for example, 'hello.sh'.
- Make the script executable using 'chmod +x hello.sh' and run it with './hello.sh'.

UNIX File System

UNIX File System Overview

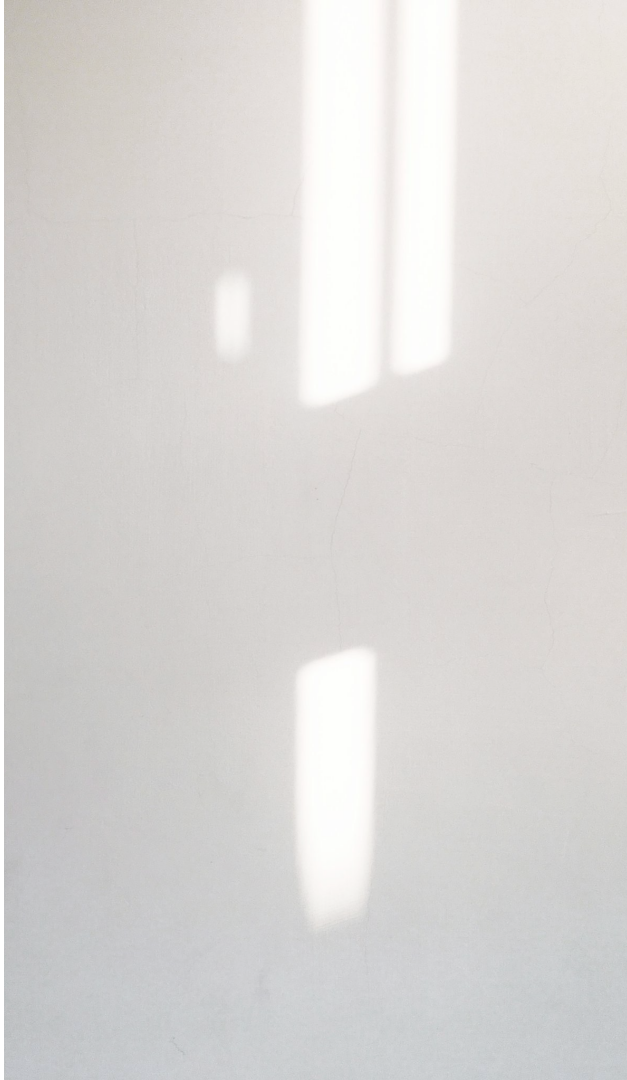
The UNIX file system is designed as a hierarchical tree structure starting from a single root directory denoted as '/'.

Directories act as containers that hold files or other directories, enabling an organized filing system. Each file or directory is uniquely identified by its path within this tree.

Mount points allow external storage devices or file systems to be integrated seamlessly into the directory tree, providing a unified view of all files and directories across devices and partitions.

File Types in UNIX

- Regular Files: Store data such as text, images, or programs; most common file type.
- Directories: Special files that act like folders to organize files and other directories hierarchically.
- Symbolic Links: Pointers or shortcuts to other files or directories, enabling flexible file referencing.
- Character Devices: Represent devices that handle data character by character, like keyboards or serial ports.
- Block Devices: Represent devices that transfer data in blocks, such as hard drives or USB drives.



Permissions

File Permissions

Permission Types

UNIX file permissions are divided into three types: read (r), write (w), and execute (x). Read allows viewing file contents, write allows modifying, and execute permits running the file as a program.

User, Group, Others

Permissions are assigned to three categories: user (file owner), group (users in the same group), and others (everyone else). Each category can have different permissions for the same file.

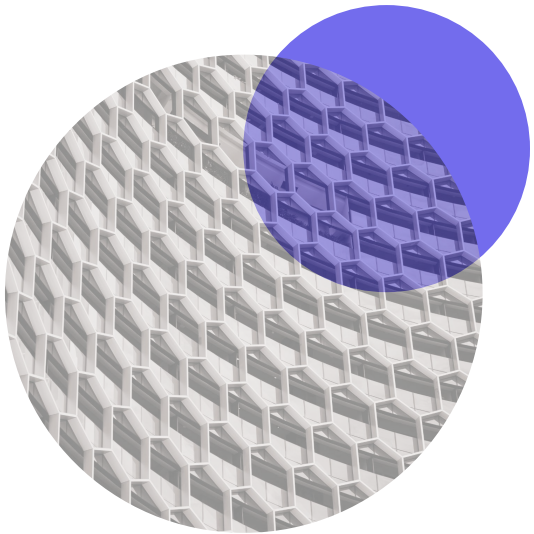
Permission Representation

Permissions are displayed as a 10-character string (e.g., -rwxr-xr--). The first character indicates file type, followed by three sets of rwx for user, group, and others, respectively.

I-node Entries Explained

An I-node (Index Node) is a key data structure in UNIX file systems that holds metadata about a file, excluding its name and actual data. It stores attributes like file size, ownership, permissions, timestamps, and pointers to data blocks. Each file has a unique I-node number used by the system to locate and manage files efficiently. This separation of metadata and file names allows for flexible file handling and fast access in UNIX.

File Related Commands



- chmod: Changes file permissions; e.g., `chmod 755 filename` grants read, write, execute to owner and read, execute to group and others.
- chown: Changes file ownership; e.g., `chown user:group filename` sets file owner and group.
- ls -l: Lists files in long format showing permissions, ownership, size, and modification date.
- stat: Displays detailed information about a file including inode, permissions, size, and timestamps.
- file: Determines and displays the file type by examining its content rather than extension.

UNIX uses three standard data streams:

Pipe Concept: Std Input, Output, Error

1. Standard Input (stdin): Default input source, usually keyboard.

2. Standard Output (stdout): Default output destination, usually terminal.

3. Standard Error (stderr): Default error message output, usually terminal.

Pipes (|) connect stdout of one command to stdin of another, enabling command chaining.

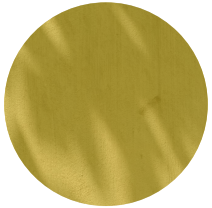
Example: ls -l | grep "^d" lists directories only.

Redirection operators manage streams separately, e.g., '2>' redirects stderr to a file.

Unix Environment: Shells Overview

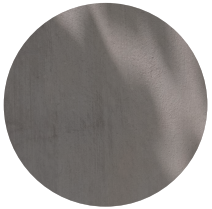
Bash Shell

Bash (Bourne Again SHell) is the most widely used UNIX shell, known for its compatibility with the Bourne shell and enhanced scripting features like command completion and history.



Korn Shell (ksh)

Korn Shell is an advanced shell that combines features of the Bourne shell and C shell, offering improved scripting capabilities, built-in arithmetic, and job control.



C Shell (csh)

C Shell provides a syntax similar to the C programming language, includes features like aliases, command history, and job control, favored for interactive use.

System Variables & Set Options

Environment & System Variables

- Environment variables store system-wide or user-specific settings.
- Common variables include PATH, HOME, USER, SHELL, and LANG.
- Use 'echo $VARIABLE_NAME' to view a variable's value.
- Set variables temporarily with 'VARIABLE_NAME=value'.
- Export variables with 'export VARIABLE_NAME' for child processes.

Viewing and Setting Shell Options

- Use 'set' command to view current shell options and variables.
- Use 'set -o' to list all shell options with their status (on/off).
- Enable a shell option with 'set -o option_name'.
- Disable a shell option using 'set +o option_name'.
- Common options include 'noclobber', 'ignoreeof', and 'xtrace'.

Unix Environment

Process Management & Background Commands

- A process is an instance of a running program in UNIX, managed by the kernel.
- The 'ps' command lists currently running processes, showing PID, TTY, TIME, and CMD.
- Use 'top' for real-time dynamic process monitoring including CPU and memory usage.
- Run commands in the background by appending '&' at the end, e.g., 'sleep 60 &'.
- Control background jobs with 'jobs', 'fg' (bring to foreground), and 'bg' (resume in background).

The cron daemon runs in the background on UNIX systems to execute scheduled commands or scripts, known as cron jobs.

Cron jobs are defined in crontab files with a syntax of five fields: minute, hour, day of month, month, and day of week, followed by the command.

Scheduler

Cron Scheduler

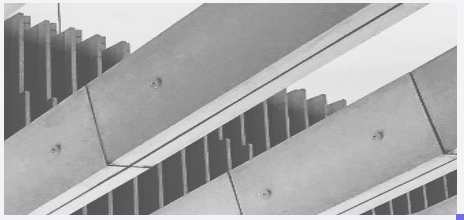
Example: To run a backup script daily at 2:30 AM, use:

```
30 2 * * * /path/to/backup.sh
```

Edit cron jobs with `crontab -e` and list them with `crontab -l`. Cron is vital for automating tasks like backups and system maintenance.

Filters

Filters: Simple & Advanced



Simple Filters: grep

The 'grep' command searches text using patterns. Example: 'grep "error" logfile.txt' finds lines containing 'error' in logfile.txt.

Simple Filters: head & tail

'head' outputs the first lines, e.g. 'head -n 5 file.txt' shows first 5 lines; 'tail' outputs last lines, e.g. 'tail -n 10 file.txt' shows last 10 lines.

Simple Filter: tr

'tr' translates or deletes characters. Example: 'tr a-z A-Z < file.txt' converts lowercase to uppercase from file.txt input.

Advanced Filters: sort

'sort' arranges lines alphabetically or numerically. Example: 'sort -n numbers.txt' sorts lines numerically from numbers.txt.

Advanced Filters: find

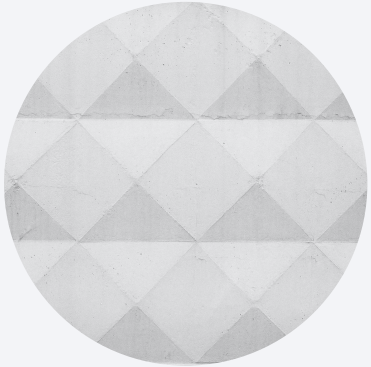
'find' searches for files by criteria. Example: 'find /home -name "*.txt"' lists all .txt files under /home directory.

Filters

Tools & Regular Expressions: sed & awk

sed: Stream Editor and Regular Expressions

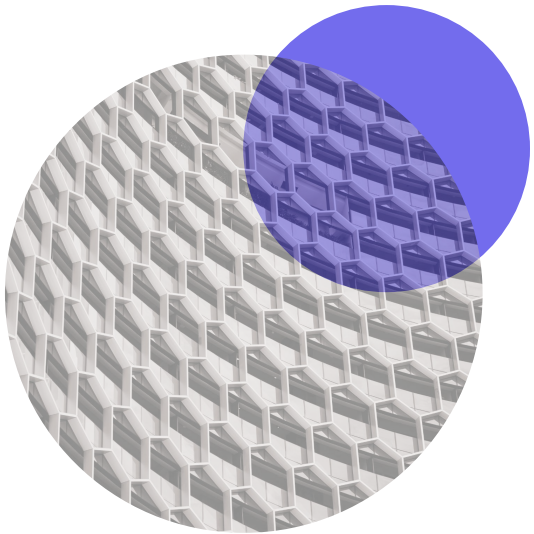
- sed is a non-interactive text editor used for parsing and transforming text streams.
- Common uses include substitution, deletion, insertion, and text replacement.
- Example: Replace 'apple' with 'orange' in a file: sed 's/apple/orange/g' filename
- Supports basic and extended regular expressions for powerful pattern matching.
- Useful for batch editing files or filtering text in pipelines.



awk: Pattern Scanning and Processing Language

- awk is a programming language designed for text processing and data extraction from structured files.
- Processes input line-by-line, splitting into fields for pattern matching and manipulation.
- Example: Print the first column of a file: awk '{print $1}' filename
- Supports arithmetic and string operations, conditions, loops, and built-in variables.
- Ideal for generating reports, extracting columns, and performing complex transformations.

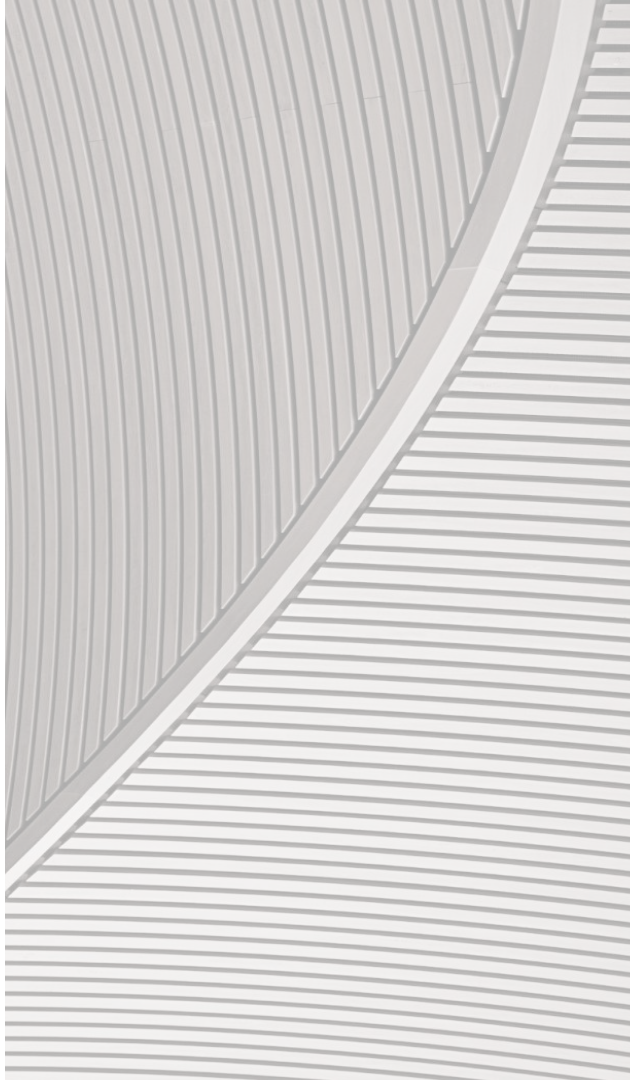
Search: File & Nested Search



- Use 'find' command to search files and directories recursively with multiple filtering options, e.g., by name, type, size.
- 'grep' command searches for specific text patterns inside files, supporting regular expressions and recursive search with '-r' option.
- Combine 'find' and 'grep' to perform powerful nested searches, e.g., find . -type f -exec grep -l 'pattern' {} \;.
- Use '-name' option in 'find' to match filenames with wildcards, e.g., find . -name '*.txt' searches all text files.
- Limit search depth with '-maxdepth' option in 'find' to control recursion levels for performance optimization.

Basic OS Commands: Process, Memory, CPU

- top: Displays real-time system summary including CPU, memory, and process info. Example: `top`
- ps: Lists currently running processes with details. Example: `ps aux` shows all processes with user info.
- free: Shows memory usage statistics including total, used, free, and swap memory. Example: `free -h` for human-readable format.
- vmstat: Provides system performance stats like processes, memory, paging, block IO, traps, and CPU activity. Example: `vmstat 2` refreshes every 2 seconds.
- df: Reports disk space usage of file systems. Example: `df -h` shows sizes in human-readable format.



Vi Editor

Vi Editor Basics

Introduction to Vi Editor

Vi is a powerful, modal text editor widely available on UNIX systems, known for its efficiency and speed in editing files without a graphical interface.

Input Mode Commands

Switch to Insert mode by pressing 'i' to start typing text. Use 'Esc' key to return to Command mode for executing commands.

Save & Quit Commands

To save changes, type ':w' in Command mode. To quit Vi, type ':q'. Combine both with ':wq' or ':x' to save and exit simultaneously.

Cursor Movement Commands

Use 'h', 'j', 'k', 'l' keys to move left, down, up, and right respectively. Other commands include 'gg' to go to start and 'G' to go to end of file.

Command Mode Essentials

Command mode lets you delete lines with 'dd', copy with 'yy', paste with 'p', and undo changes with 'u' for efficient text manipulation.

Practical Tip

Practice switching between modes and using simple commands to speed up editing. Vi is ideal for quick file edits on remote UNIX servers.

Shell Programming: Variables & Commands

Shell Variables

Shell variables store data within a script or session. They are assigned without spaces using VAR_NAME=value. Example: NAME="John" assigns the string John to variable NAME. Access with $NAME.

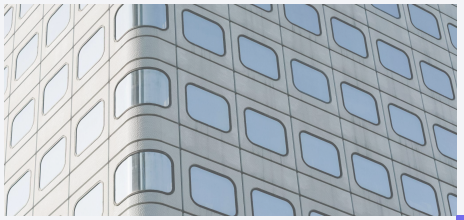
Environment Variables

Environment variables are global and inherited by child processes. Common ones include PATH (directories to search for commands) and HOME (user's home directory). Set with export, e.g., export PATH=$PATH:/new/path.

Basic Shell Commands

Shell scripts use commands like echo to display text, read to capture input, and test ([ ]) for conditions. Example: echo "Hello $NAME" prints Hello followed by the NAME variable's value.

Shell Programming: Control Structures



Conditional Execution with if Statements

The 'if' statement allows execution of commands based on conditions. Syntax: if [condition]; then commands; fi. Example: if [$age -ge 18]; then echo 'Adult'; fi.

Using the test Command

The 'test' command evaluates expressions like string comparison, file checks, and integers. Example: test -f filename && echo 'File exists'. It is often used within 'if'.

Case Statements for Multiple Conditions

The 'case' statement handles multiple possible values of a variable efficiently. Syntax: case $var in pattern1) commands ;; pattern2) commands ;; esac. Example: case $color in red) echo 'Stop';; green) echo 'Go';; esac.

Loops: while and until

'while' executes commands repeatedly while a condition is true; 'until' runs until a condition becomes true. Example: while [$count -le 5]; do echo $count; ((count++)); done.

Control Flow: break and continue

'break' exits a loop prematurely, 'continue' skips the current iteration. Useful for controlling loop behavior based on dynamic conditions.

Shell Programming: Functions & Arrays

Shell Functions

- Functions group commands for reuse, improving script modularity and readability.
- Declared using syntax: function_name() { commands; } or function function_name { commands; }
- Called by simply using function_name in the script.
- Example:
- myfunc() {
- echo "Hello from function"
- }
- myfunc # Executes the function

Arrays in Shell

- Arrays hold multiple values indexed starting at 0 for efficient data handling.
- Declared with syntax: array_name=(value1 value2 value3)
- Access elements using: ${array_name[index]}, e.g. ${array_name[0]}
- Example:
- colors=(red green blue)
- echo ${colors[1]} # Outputs 'green'

Use Case

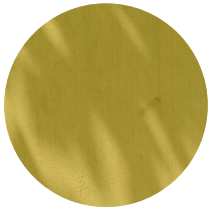
Use Case: File Operations with Shell Script

Read File Line by Line

Use a while loop with 'read' to process each line:

```bash
while IFS= read -r line; do
  echo "$line"
done < filename.txt
```

This reads and prints each line from 'filename.txt'.



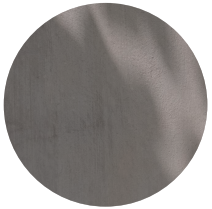
Concatenate & Split Files

Concatenate files using 'cat':

```bash
cat file1.txt file2.txt > combined.txt
```

Split large files with 'split':

```bash
split -l 1000 largefile.txt part_
```



Touch File & Check Size

Create or update a file timestamp with 'touch':

```bash
touch newfile.txt
```

Check file size with 'stat':

```bash
stat -c %s filename.txt
```

Use Case

Use Case: Text Processing & Commands

Record & Word Count

Use 'wc' command to count lines, words, and characters. Example: 'wc -l filename' counts lines; 'wc -w filename' counts words.

AWK and SED Operations

AWK processes text line-by-line for pattern scanning and reporting. SED is a stream editor for text substitution. Example AWK: 'awk '{print $1}' file' prints first column. Example SED: 'sed 's/old/new/g' file' replaces text.

Head & Tail Commands

'head' displays the first lines of a file; 'tail' shows the last lines. Example: 'head -5 filename' shows first 5 lines, 'tail -10 filename' shows last 10 lines.

Use Case

Use Case: SFTP & DB Connection

Using SFTP for Secure File Transfer

- SFTP (Secure File Transfer Protocol) encrypts data during transfer ensuring confidentiality.
- Basic SFTP commands: `sftp user@host` to connect, `put filename` to upload, `get filename` to download.
- Example: `sftp user@192.168.1.10` then `put report.txt` uploads the file to remote server.
- Use batch mode with `-b` option for automated scripted transfers without manual intervention.
- SFTP sessions can be secured with SSH keys for passwordless authentication.

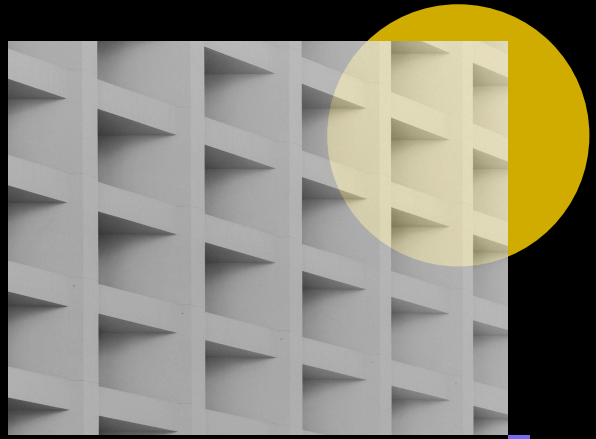
Connecting to Database & Executing DDL/DML

- Shell scripts can connect to databases using command line clients like `mysql` or `psql`.
- Example to run SQL commands in MySQL: `mysql -u user -p password -e "SQL Query"`
- DDL (Data Definition Language) commands: CREATE, ALTER, DROP tables to modify schema.
- DML (Data Manipulation Language) commands: SELECT, INSERT, UPDATE, DELETE to manage data.
- Sample script snippet: `mysql -u root -p -e "CREATE DATABASE testdb; USE testdb; CREATE TABLE users (id INT, name VARCHAR(50));"`

Conclusion

Summary and Next Steps

- UNIX provides a robust multiuser environment with essential commands and scripting.
- File system knowledge including permissions and I-nodes is key to managing UNIX effectively.
- Shell scripting automates repetitive tasks and enhances system control.
- Filters like grep, sed, and awk enable powerful text processing.
- Practice with real-world use cases and explore process management and scheduling for deeper learning.



Thank you