GNU+Linux with Shell Scripting

PRESIDIO Academy Batch 2025

Training

Plan

- Many people think of the *shell* (BASH, ZSH) when they think of GNU+Linux, but it’s more than a shell.
- In fact, 60% of the training is going to be *not* about the shell, but the internals of the system.
- This training follows a *principles-first* approach, ensuring you are able to answer the “why”, “how”, and “what” questions of the most fundamental concepts.
- But don’t worry. We’re going to have as much *hands-on/practical work* as we learn all the theoretical concepts.

Training

Plan (contd.)

- This training is happening across 4 days.
- First 3 days is about fundamentals of GNU+Linux. The fourth last day will be about shell scripting.

Training

Syllabus

- **Day-1**: Foundations Of GNU+Linux, History, UNIX And POSIX, Kernel Components, Distributions, Shell Types, Boot Process, File System Hierarchy, man Pages
- **Day-2**: System And Process Management, Users And Groups, Filesystem Operations, Process And Daemon Control, Privilege Management, Logs, Security Basics
- **Day-3**: Networking And Software Management, Package Managers, wget And cURL, File Transfer Tools, SSH Administration, Network Interfaces And Routing, Firewalls
- **Day-4**: Shell Scripting Essentials, BASH Basics, Quoting And Expansion, Control Flow, Redirection And Subshells, Globbing And Regex, Best Practices

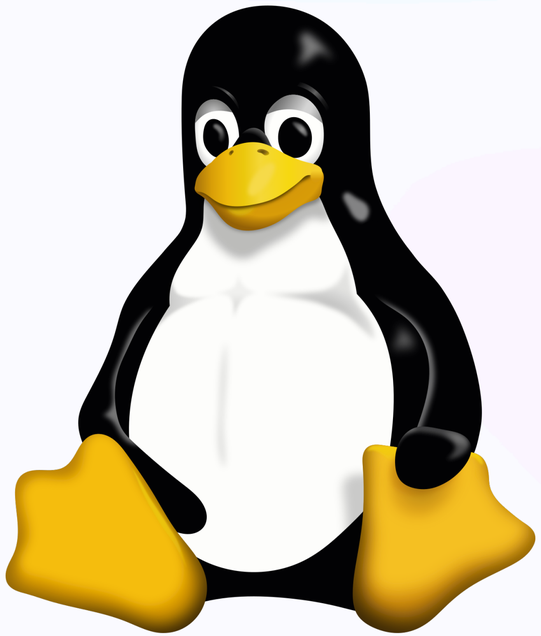
Your Trainers

- Pavithra P
- Shrish Mariappan
- Sakthi Santhosh
- Sugavanesh Murugesan

A General Misconceptions

- Many believe Linux is an operating system, but it’s the kernel, the core part that interacts with hardware. The full OS is typically “GNU+Linux,” since it’s the combination of the Linux kernel with GNU tools that makes a usable system.

+



History of GNU+Linux

Important Events and People

AT&T Bell Laboratories

~10,500

Number of Lines of Code in the Linux Kernel at its Inception

~3,00,00,000

Number of Lines of Code in the Linux Kernel at Present

~15,000

Developers Contribute to the Kernel

~70,000

Commits Per Year

~80%

Smartphones Run on Linux (via Android)

~70%

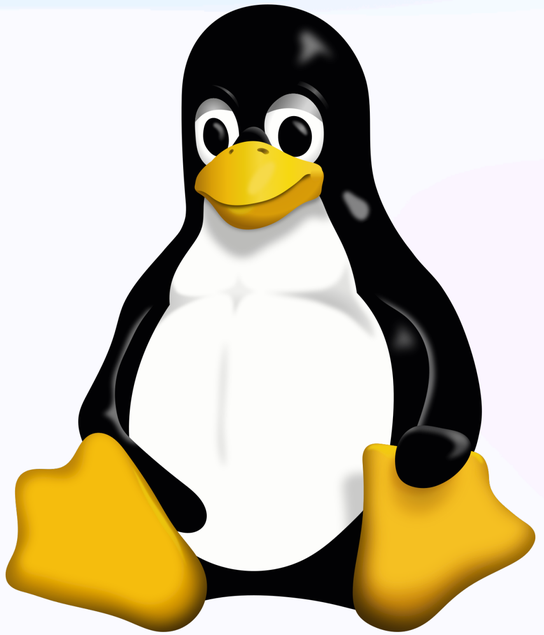
Servers on the Cloud Run of Linux

~70%

Servers on the Cloud Run of Linux

GNU+Linux is Everywhere

+



UNIX and POSIX

Principles and Standards

UNIX

And Its Philosophies

- UNIX is an operating system first created at Bell Labs in 1969 by Ken Thompson, Dennis Ritchie, and others.
- Designed to be simple, powerful, and flexible, it introduced the idea that “everything is a file.”
- Key features: multitasking, multiuser support, and tools that can be combined in powerful ways.

Do one thing and do it well.

Build complex solutions by connecting simple tools.

Use text as a universal interface.

POSIX

OS Standard

- POSIX stands for Portable Operating System Interface, a set of standards developed in the 1980s.
- Its goal: Make sure software written for one UNIX-like system will work on others with minimal changes.
- POSIX defines rules for how the system should behave, including command-line tools, file operations, and how programs run.
- Almost all modern UNIX-like systems, including Linux, BSD, and macOS—follow POSIX standards.

POSIX is the *guide*,
UNIX is the *implementation*.

UNIX and POSIX

Important Distinctions

- When I say UNIX, I mean an operating system family with its roots in AT&T Bell Labs, and later a trademark owned by The Open Group.
- When I say POSIX, I mean compliance with a checklist of behaviors and interfaces defined by IEEE that any system can implement.
- Not all POSIX systems are UNIX systems. Linux is POSIX-compliant but not UNIX.
- But all certified UNIX systems are, by requirement, POSIX-compliant.

UNIX

Operating Systems Born from It

FreeBSD

Android

iOS

macOS

What is an OS?

Mechanisms and Policies

- **OS = Virtualizer of Resources**
  - Hardware is limited (CPU, memory, I/O, devices).
  - OS gives each process the illusion of having the whole system.
- **Mechanisms vs. Policies**
  - Mechanisms (the how): Context switch, paging, scheduling and interrupts.
  - Policies (the what): Which process runs, how memory is divided, who gets priority.

Terminologies

The Big Picture

Too Many Terms

Kernel

The Big Picture

Too Many Terms

Desktop Environment

Distributions

Window Manager

Kernel

Init System

Flavors

Release Cycle

Package Manager

Linux Ecosystem

Big Picture

- **Linux Kernel** = Heart of the OS
- **Distros** = Kernel + GNU Tools + Package Management + Userland Utilities
- **Flavors**: Modifications over distributions for specific purposes.

Linux Ecosystem Terminologies

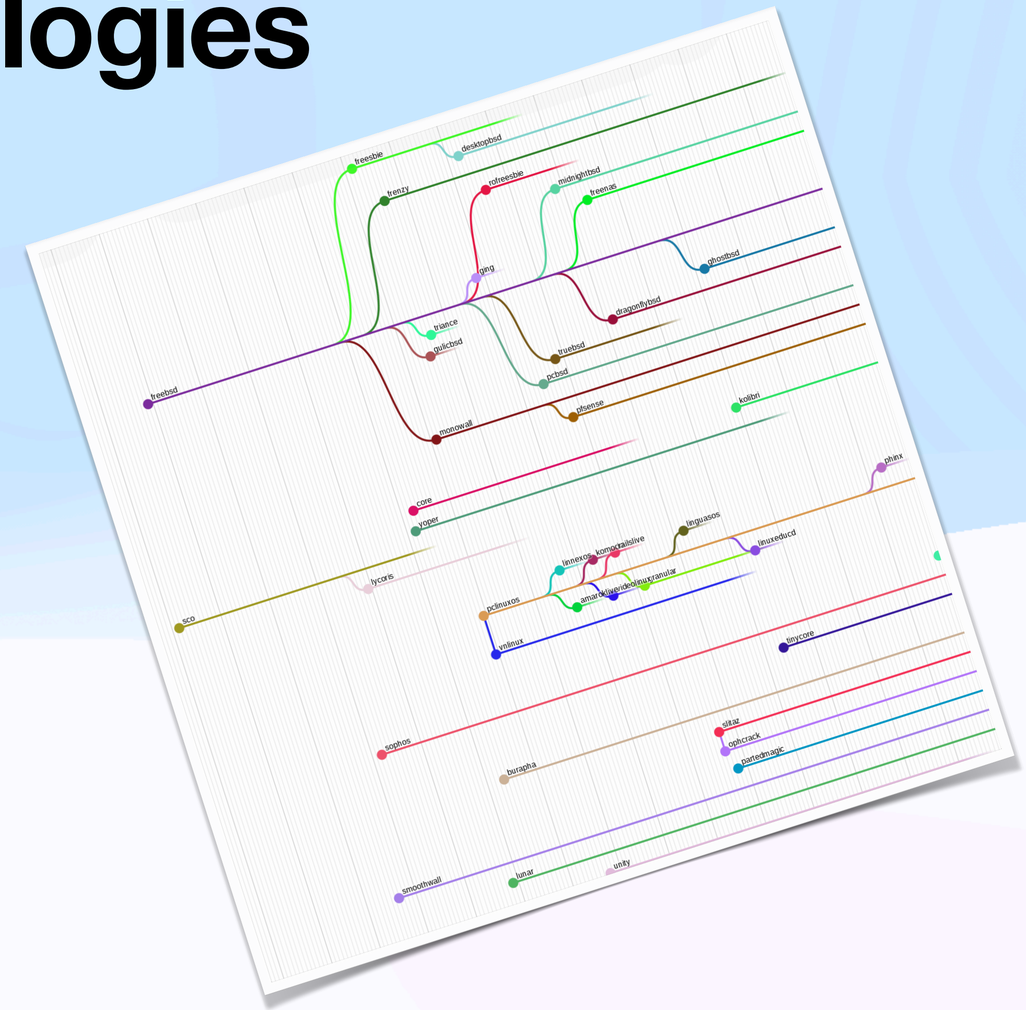
Distribution

- A Linux distribution (distro) is a complete OS built around the Linux kernel.
- **Includes:**
  - Package Manager (`apt`, `yum/dnf`, `pacman`, `nix`)
  - Core Utilities & Libraries (BASH and Other Shell Utilities)
  - Installer, System Tools, GUI/DE
  - **Examples**: Ubuntu, Fedora, Arch, Debian, RHEL

Linux Ecosystem Terminologies

Mother of All Distributions

- Debian: Ubuntu, Mint and Kali Linux
- Arch: Manjaro, Arco and EndeavorOS
- RedHat Enterprise Linux: CentOS, Fedora



https://distrowatch.com/images/other/distro-family-tree.png

Linux Ecosystem Terminologies

Flavors

- **Community vs Enterprise**
  - Ubuntu, Debian, Arch → Community
  - RHEL, SUSE, Ubuntu Pro → Enterprise
- **Lightweight vs Full-featured**
  - Alpine, BusyBox → Minimal
  - Ubuntu, Fedora → Desktop-ready
- **General-purpose vs Specialized**
  - Kali Linux (security), Raspbian (IoT), Pop!_OS (dev/desktops)

Linx Ecosystem Terminologies

Release Cycle

- **Fixed Release (Stable)**
  - Debian Stable, RHEL
  - Predictable updates, focus on stability.
- **Rolling Release**
  - Arch, openSUSE Tumbleweed
  - Continuous updates, latest software.
- **LTS (Long Term Support)**
  - Ubuntu LTS (5 years support)
  - Balance between stability & modernity.

Desktop Environments

Most Popular Ones

LXQt

KDE Plasma

elementary OS

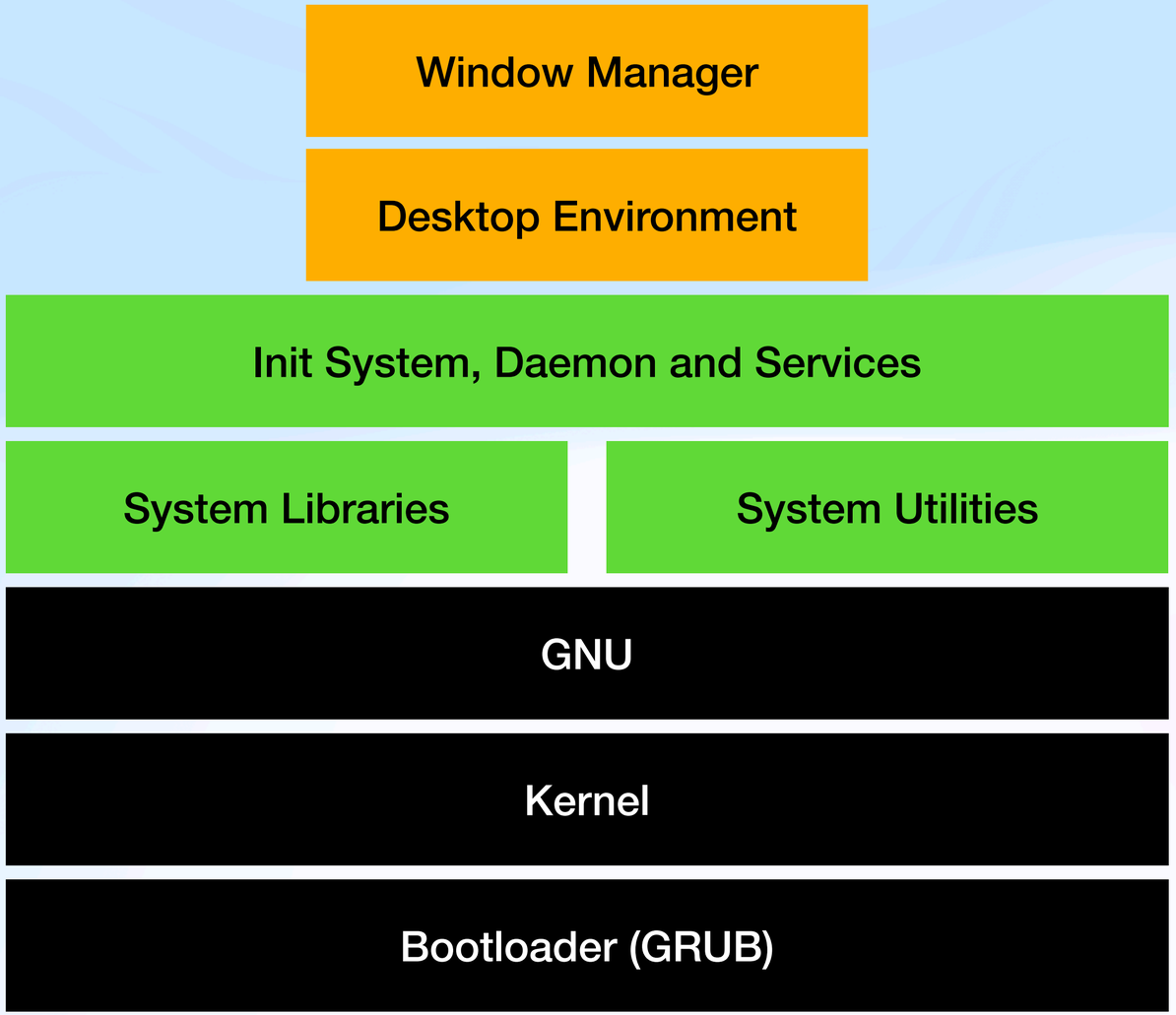
XFCE

GNOME™

https://apps.kde.org/

Linux Ecosystem Terminologies

Your Distribution Stack



>1000

Distributions with ~500 actively maintained ones.

Mid-training Summary

- POSIX is the *guide*, UNIX is the *implementation*.
- Linux is an kernel, GNU+Linux is an OS.
- A Linux-based OS/Distribution contains of:
  - Bootloader
  - Kernel and GNU
  - System Libraries and Utilities
  - Init System, Daemon and Services
  - Desktop Environment

Console, Shell and Terminal

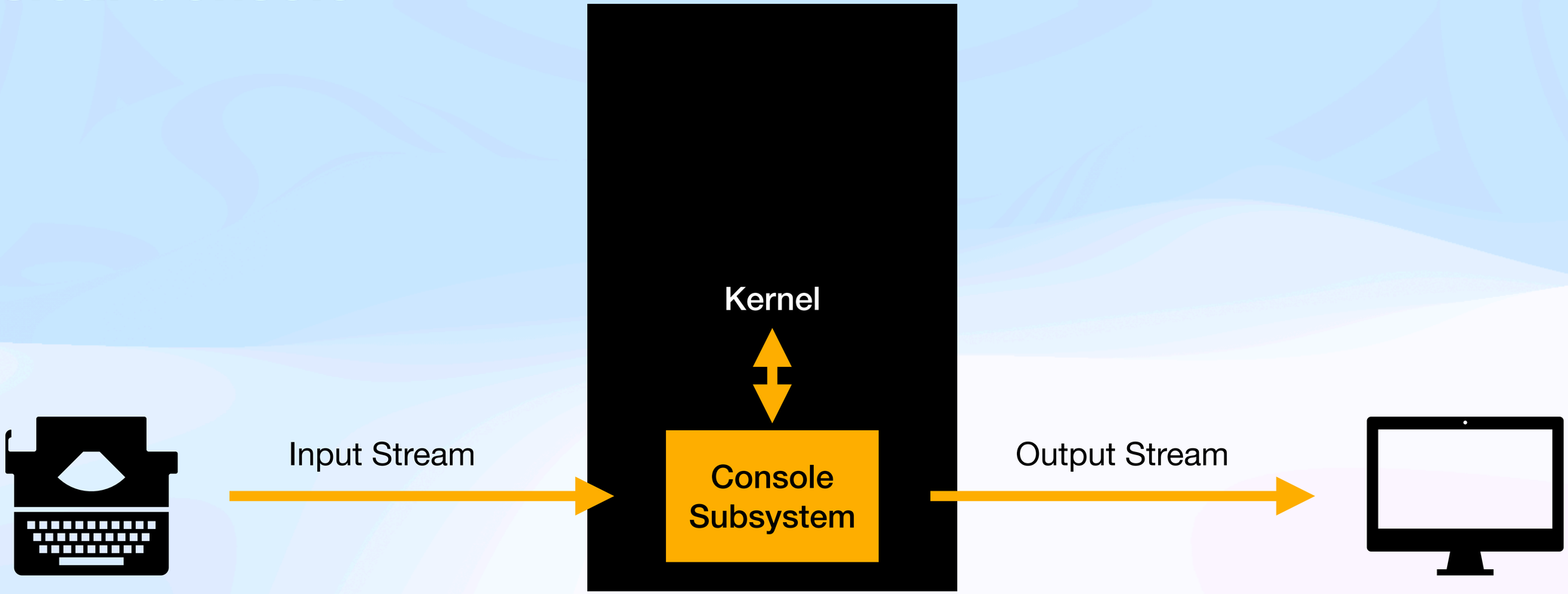
Console

Interface Between Kernel and You

- A text-based interface to interact with the Linux kernel.
- Historically tied to physical terminals (keyboard + monitor).
- Provides direct access without needing a graphical server.
- Represents the lowest-level user interface available.

Console Subsystem

Physical Console



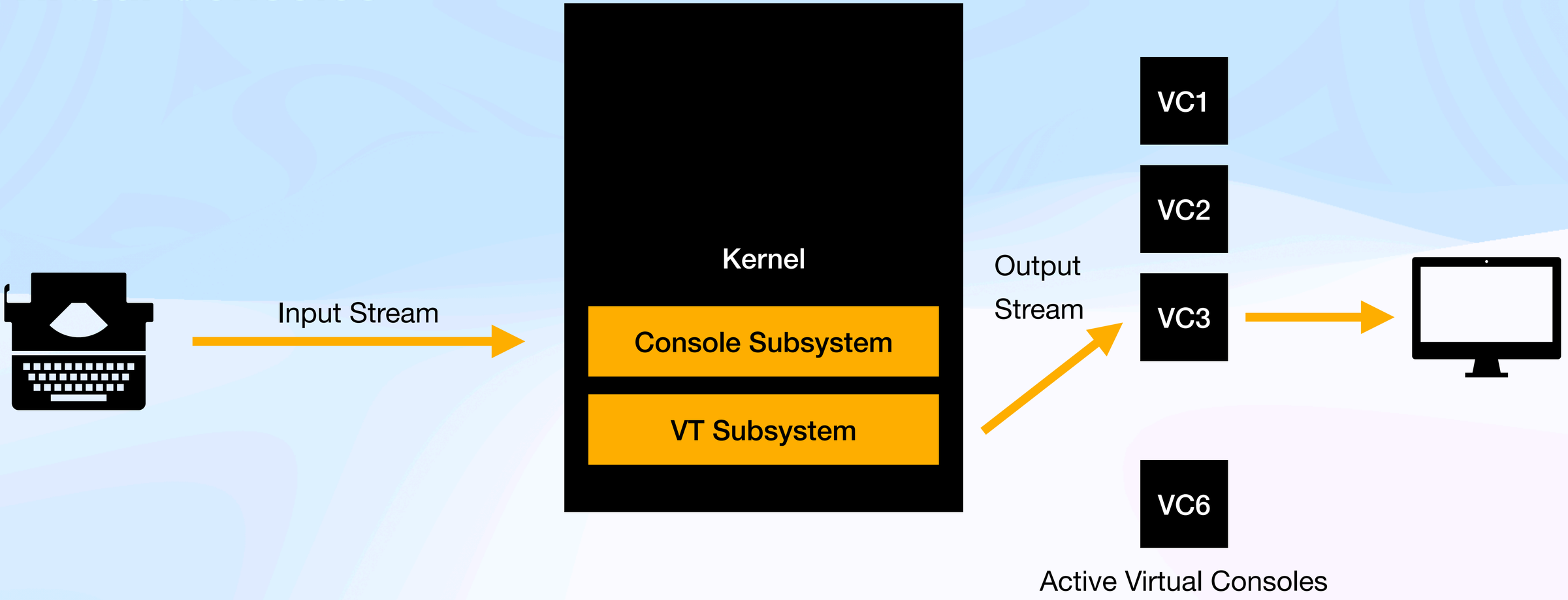
Console

History

- Early Unix systems used teletypes (TTYs).
- `tty` command in Linux still shows the terminal device.
- Physical consoles evolved into virtual consoles in Linux.
- You can switch between them using `Ctrl+Alt+F1` to `F6`.

Console Subsystem

Virtual Consoles



Console

Types

- **Physical Console**: Directly attached monitor and keyboard.
- **Virtual Consoles**: Software-provided consoles (/dev/tty1, /dev/tty2, …).
- **Serial Consoles**: Access via serial port (common in servers and embedded).
- **Remote Consoles (pseudo)**: Accessed via SSH, telnet, or remote management systems.

Shell

Shell

- A command-line interpreter between user and kernel.
- Accepts text input, executes commands and displays output.
- Called “shell” because it wraps around the kernel.
- Runs on top of the console or terminal.

Shell

Types

- **`sh`**: The original Bourne shell.
- **`bash`**: Most common, default in many distros.
- **`zsh`**: Popular for developers (autocompletion, plugins).
- **`fish`**: User-friendly, interactive.
- **`nushell`**: Lightweight, POSIX-compliant (used in scripts).

Shell

Components

- **Prompt**: Indicates shell readiness ($, #).
- **Parser**: Splits input into command + arguments.
- **Execution Engine**: Runs programs using `fork` + `execve`.
- **Environment**: Variables like `PATH`, `HOME`.
- **Scripting Layer**: Control flow, loops, functions.

Shell

Console without It

- The shell is the command interpreter; without it, the console is just a raw input/output channel.
- Keystrokes would be sent directly to the system as uninterpreted characters, not as commands.
- You cannot run programs, use redirection, or even navigate the filesystem, only the kernel/system daemons may still print their messages.
- No shell prompt after you login, just plain screen.

Shell

What’s Next

This is all for now. We’ll cover in-depth about BASH in the last day of this training.

Terminal

Terminal

- Originally hardware devices (teletypes, VT100).
- Today, they are software programs (terminal emulators).
- Provides text input/output interface for the shell.
- **Examples**: GNOME Terminal, Konsole (from KDE), Xterm, Alacritty.

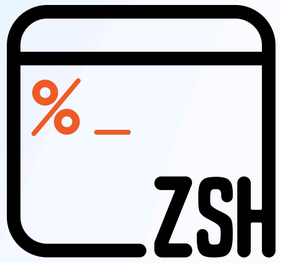
Terminal

- Displays output from programs.
- Sends keystrokes to the shell.
- **Supports:**
  - Scrollback Buffer
  - Colors & Escape Sequences
  - Copy–paste
- But it does not execute commands.

Terminoligies

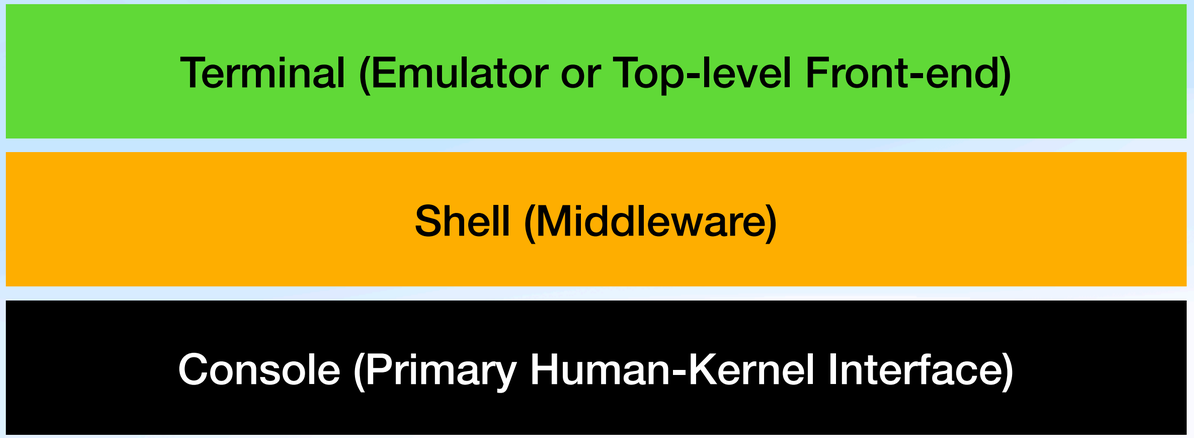
Summary

- **Console**: The physical (or virtual) device providing text I/O with the system exposing the kernel for operations.
- **Shell**: The command interpreter program (like bash, zsh, fish).
- **Terminal**: An abstraction (software or hardware) that handles input/output streams.



Console

Stack



File System Hierarchy

File System Hierarchy

Background

- The filesystem hierarchy is the standardized directory structure followed by GNU/Linux.
- Defined by FHS (Filesystem Hierarchy Standard) maintained by the Linux Foundation.
- Everything in Linux is a file: binaries, text, devices, even sockets.
- Unlike Windows, Linux doesn’t rely on drive letters (`C:`, `D:`). Instead, everything starts at `/`.

File System Hierarchy

Important Directories

- **`/boot`**: Kernel, initrd, Bootloader Files
- **`/dev`**: Device Files
- **`/etc`**: System Configuration Files
- **`/var/log`**: System Logs
- **`/bin`**, **`/sbin`**, **`/usr/bin`**, **`/usr/sbin`**: Essential and Non-essential User Binaries
- **`/proc`**: Process Related Files

Boot Process

The Boot Process

High Level Stages

BIOS/UEFI → Bootloader (GRUB) → Load Kernel (with `initramfs`) → Mount FS → Run `systemd` → Login Screen

Boot Process

The Firmware

- Runs POST (Power-On Self Test)
- Initializes CPU, memory, and peripherals.
- Reads boot device order.
- Loads bootloader from disk (MBR or EFI partition).

Boot Process

Bootloader Stage

- Located in MBR or EFI partition
- Examples: GRUB, LILO, systemd-boot
- **Responsibilities:**
  - Present boot menu.
  - Load kernel + `initramfs` into memory.
  - Pass control to kernel.

Boot Process

Kernel Initialization

- Decompresses itself into memory.
- Initializes device drivers.
- Mounts root filesystem (with help of `initramfs`).
- Starts the first process: `/sbin/init` (PID 1).

Boot Process

Init System

- First process (PID 1)
- Historically: `SysVinit` (simple scripts)
- Modern: `systemd` (parallel service startup)
- **Responsibilities:**
  - Mount remaining filesystems.
  - Start daemons (`sshd`, networking, logging).
  - Reach target run level (`multi-user/graphical`).

Boot Process

User Space Initialization

- Login prompts started (TTYs, display manager).
- User enters credentials.
- Shell/desktop environment launched.

man Pages

Shell Keyboard Bindings

Summary

Here’s What you Learnt

- History
- UNIX and POSIX Principles
- Terminologies in the Linux Ecosystem
- Console, Shell and Terminal
- File System Hierarchy
- The Boot Process
- man Pages