[Overview] k1 — Orientation: What “Linux” Means Across Distros □□

1.1 Distributions & families

What “Linux” usually means in practice

When people say “Linux,” they often mean an entire operating system stack:

- **Linux kernel** (the core: hardware, processes, memory, networking)
- **Userland tools** (GNU coreutils like `ls`, `cp`, `grep`, `tar`, etc.)
- **Init/service manager** (commonly **systemd** today)
- **Package manager + repositories** (how software is installed/updated)
- **Default configs** and security policies

A **distribution (distro)** is a curated bundle of those pieces with a particular philosophy (stability vs newness), defaults, and packaging.

The two big server families you’ll meet most

1. **Debian family**
   - **Debian** (very stability-focused)
   - **Ubuntu** (based on Debian; popular; Ubuntu **LTS** is common on servers)
2. **RHEL family** (Red Hat Enterprise Linux–compatible)
   - **Rocky Linux**, **AlmaLinux** (common on “enterprise-style” servers)
   - Historically CentOS filled this role

 What's shared across both: filesystem concepts, permissions, SSH, networking basics, processes, most command-line tools, and the general "shape" of running a web server.

1.2 What differs (and why it matters)

Here's the "difference map" you'll keep using throughout the course:

1. **Package management**
   - Debian/Ubuntu: `apt`
   - Rocky/Alma: `dnf` (and sometimes legacy `yum`)
   - Practical impact: different commands, sometimes different package names, and sometimes different "recommended" installation methods.
2. **Release cadence (how fast things change)**
   - Debian stable / RHEL-like: slower-moving, long lifecycle
   - Ubuntu LTS: stable, but often newer than Debian stable
   - Practical impact: versions of PHP, Nginx, Node, etc. available via default repos.
3. **Security frameworks & defaults**
   - RHEL family: **SELinux** is a major factor
   - Ubuntu: **AppArmor** is common
   - Practical impact: the same config can "work" on one distro but get blocked by MAC policies on another until you learn to diagnose it.
4. **Default tooling and conventions**
   - Firewall tools: often **UFW** (Ubuntu) vs **firewalld** (RHEL family)
   - Log locations and config paths are *mostly* similar, but defaults can differ.
   - Practical impact: you'll learn to *find* the right file/service rather than memorize one path.

 Key mindset: **Learn the core once, then learn the "translation layer."** This course is structured exactly that way.

1.3 Your working environment (local vs server)

You'll typically interact with Linux in one of these modes:

1. **Local Linux** (installed on your machine)
   - Best if you want Linux as your daily driver.
   - Not required for learning server ops.
2. **VMs (recommended for learning)**
   - Run Debian/Ubuntu/Rocky/Alma as virtual machines and break/fix safely.
   - Works great with snapshots.
3. **Remote servers (VPS/cloud)**
   - Realistic: public IP, SSH, DNS, firewall, actual deployment constraints.
   - Higher stakes if you misconfigure something.
4. **Containers**
   - Great for reproducible app environments (Node/React, services).
   - Not a full “OS admin” experience by itself (containers share the host kernel).

My recommended practice setup for you (web dev + WordPress)

- **One VM** as your “sandbox server”
- **One cheap VPS later** (optional) to practice real-world networking/TLS
- Add **containers** once you start Node/React workflows

1.4 Safety & learning workflow

The “don’t regret it later” rules

1. **Always have a rollback**
   - Use VM snapshots before major changes.
   - Keep copies of config files before editing (even a simple `cp file file.bak`).
2. **Make changes intentionally**
   - Change *one thing at a time*.
   - Keep a small change log for yourself (what you did + why).
3. **Practice in layers**
   - First: command line + files + permissions
   - Then: services + logs
   - Then: networking + firewall
   - Then: web stack + TLS

Tools you’ll likely use

- **SSH** for remote access
- A terminal editor (**nano** or minimal **vim** skills)
- A way to snapshot/clone environments (VM snapshots; later Docker images)

Mini roadmap: what’s next □□

1. **k2** makes you comfortable in the shell (portable across distros).
2. **k3** teaches filesystems + permissions (critical for WordPress security and debugging).
3. Later chapters introduce the **translation layer** (APT vs DNF, UFW vs firewalld, AppArmor vs SELinux).

Quick check-in (optional, to tailor future lessons)

What will you use first for practice: **VM**, **a VPS**, or **WSL/macOS terminal + remote server**?

Revision #3
Created 2026-04-04 23:18:11 UTC by art10m
Updated 2026-04-06 20:23:19 UTC by art10m