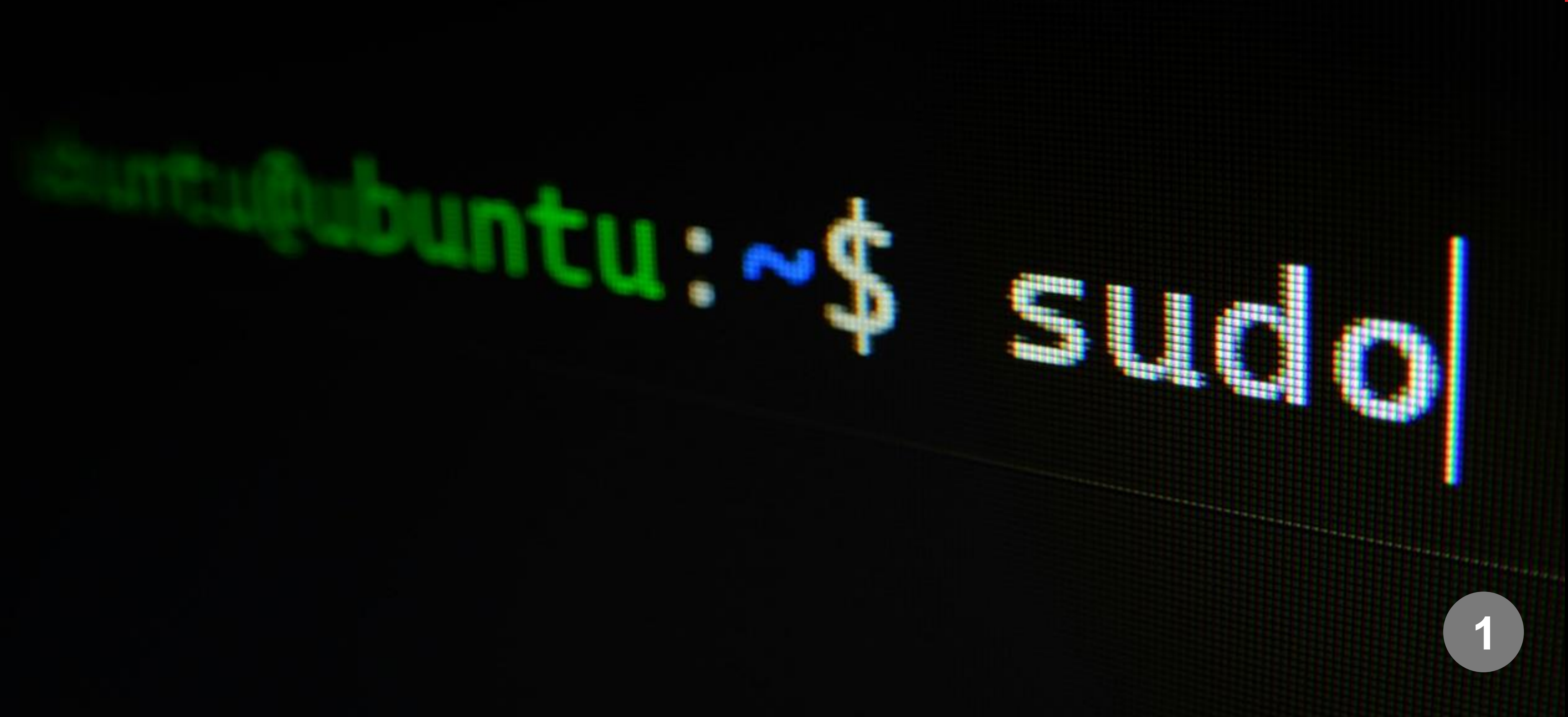
Linux: System Administration



LOGISTICS

Class Hours:

- Instructor will set class start and end times.
- There will be regular breaks in class.

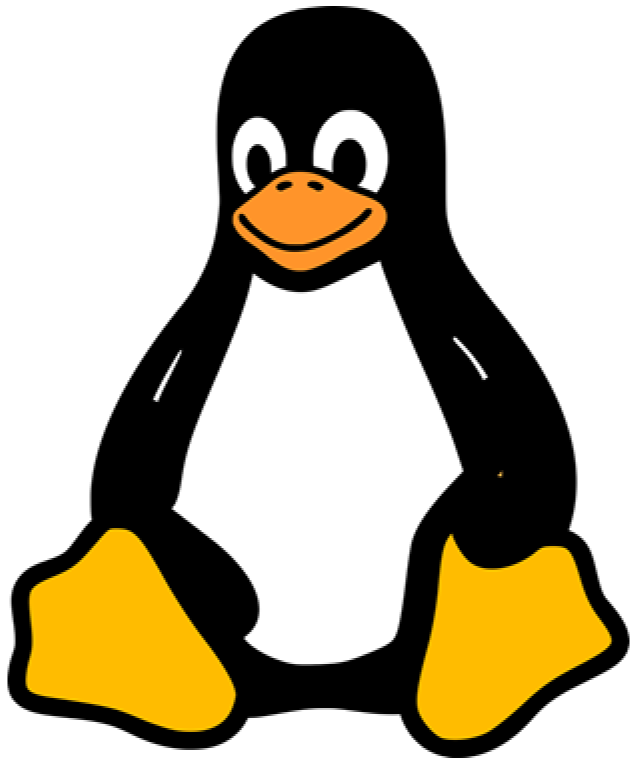
Telecommunication:

- Turn off or set electronic devices to silent (not vibrate)

Learning:

- Run the commands with the instructor as the slides are presented to you
- Ask questions and participate

Todays Objectives



1. **Install and manage installations with package manager and manually**
2. **Understand Network configurations and monitoring**
3. **Maintain Services running on your system**
4. **Manage Disk and file systems**
5. **Advanced Process Management**
6. **Common configuration files (sudoers, profiles, logrotate)**

Linux: Package Management

Package management allows you to **install, update, and remove software** safely and efficiently.

It keeps systems **consistent, secure, and up to date**, ensuring dependencies are handled automatically.

Most distributions use tools like **dnf**, **yum**, or **apt** — mastering them helps you manage everything from servers to desktops with confidence.

Knowing how to query, verify, and roll back packages lets you **control change** and **maintain stability** across environments.

💡 A good admin doesn't just use software — they understand how it's installed, updated, and maintained.

Linux: Package Management

📦 Understanding Package Managers in Linux

Linux uses **package managers** to install, update, and remove software while handling dependencies automatically.

They come in **two layers** — high-level tools and low-level tools.

Linux: Package Manager

🧩 High-Level Package Managers

Front-end tools that handle dependencies, repositories, and updates automatically:

Distribution	**Tool**	**Description**
RHEL / CentOS / Fedora	`dnf`, `yum`	High-level front-ends for RPM packages
Debian / Ubuntu	`apt`, `apt-get`	High-level front-ends for DEB packages
Arch Linux	pacman	Unified tool that manages `.pkg.tar.zst` packages and dependencies

💡 **Best practice:** use the *high-level manager* (`dnf`, `apt`, or `pacman`) —
it ensures dependency resolution and system consistency automatically.

Linux: Package Manager

⚙️ Low-Level Package Managers

Work directly with package files (no dependency resolution):

Package Type	**Tool**	**Example Command**
.rpm	`rpm`	sudo rpm -ivh package.rpm
.deb	`dpkg`	sudo dpkg -i package.deb

💡 **Best practice:** use the *high-level manager* (`dnf`, `apt`, or `pacman`) — it ensures dependency resolution and system consistency automatically.

Linux: Package Manager

📦 Working with DNF – Basic Commands

`dnf` is the **package manager** used on RHEL, CentOS, and Fedora systems.
It handles installation, updates, and dependency resolution automatically.

```
# Search for a package
dnf search httpd

# View detailed info about a package
dnf info httpd

# Install a package
sudo dnf install httpd –y # -y installs without prompting
```

💡 **Tip:**

- Always use `sudo` when installing or modifying packages.
- `dnf` automatically pulls in required dependencies from configured repositories.

Linux: Package Manager

📋 Listing Installed Packages with DNF

You can view all packages installed on your system using `dnf list installed`. Combine it with commands like `grep` or `head` to filter and manage large outputs.

```
# List all installed packages
dnf list installed

# Show only the first 10 entries
dnf list installed | head

# Search for a specific package
dnf list installed | grep nginx
```

💡 **Tip:**

- Combine `grep` with `dnf list installed` to quickly verify if a package is present.
- Useful for audits, troubleshooting, or checking version consistency across systems.

Linux: Package Manager

🧩 Managing Package Versions with DNF

Sometimes you need a **specific version** of a package — for compatibility, stability, or rollback purposes.
dnf makes it easy to view and install older or alternate versions from your repositories.

```
# Search available versions of a package
dnf list httpd --showduplicates
```

You'll see output like:

```
httpd.x86_64  2.4.57-1.el9_2   appstream
httpd.x86_64  2.4.58-1.el9_3   appstream
```

To install a specific version:

```
sudo dnf install httpd-2.4.57-1.el9_2
```

💡 **Notes:**

- Always match the full version–release string (e.g., `2.4.57-1.el9_2`).
- Downgrading or pinning versions can prevent breakage when other packages depend on older libraries.
- Use `dnf versionlock` (from `dnf-plugins-core`) if you need to **lock** a version.

Linux: Package Manager

🔄 Downgrading or Removing Packages

Sometimes a newer version introduces bugs or breaks compatibility.
You can **downgrade** or **uninstall** packages with DNF safely and cleanly.

```
# Downgrade to a previous version
sudo dnf downgrade httpd

# Or specify an exact version
sudo dnf install httpd-2.4.57-1.el9_2

# Remove a package
sudo dnf remove httpd
```

💡 **Note:**

- DNF automatically handles **dependencies and conflicts** when downgrading or removing.
- Using **low-level tools** like `rpm` makes this harder — they don't resolve dependencies or maintain system consistency.
- Always verify version availability before downgrading (`dnf list httpd --showduplicates`).

Linux: Package Manager

🧠 Understanding Updates vs Upgrades

Before managing software, it's important to know what each action means:

Term	**Scope**	**Description**
Update	*Package-Level*	Installs the **latest version** of existing packages from current repositories (bug fixes, security patches, small feature updates).
Upgrade	*System-level*	Moves the system to a **new release version** (e.g., Fedora 38 → 39 or RHEL 8 → 9). Includes new repositories and dependencies.
Patch / Security Update	*Targeted*	Fixes specific vulnerabilities or stability issues — often applied selectively or automatically.

💡 **Think of it this way:**

- **Update** = refresh what you already have.
- **Upgrade** = move to a newer version of the OS.
- Always review release notes before performing full system upgrades.

Linux: Package Manager

🔍 Checking for Package Updates & Upgrades

Regularly checking for package updates keeps your system **secure and current**, though updates can occasionally introduce **breaking changes**.

Always **stage and test** updates before deploying them to production.

```
# List available package updates and OS upgrades
dnf check-update

# Show information about security updates only
dnf updateinfo list security
```

💡 **Tip:** Reviewing updates first helps you plan maintenance windows and avoid unplanned restarts. If the OS release metadata changes, **DNF will also alert you** that a new system release (upgrade) is available.

Linux: Package Manager

⬆️ Updating Packages with DNF

You can update all packages or just specific ones when newer versions are available.

```
# Update a specific package
sudo dnf update -y httpd

# Verify the version after update
dnf info httpd
```

💡 **Tip:**

- -y auto-confirms the update prompt (use with caution).
- Not every update is safe — test in a staging environment to catch regressions early.

Linux: Package Manager

⚠️ `dnf update` vs `dnf upgrade`

Both commands bring your system **up to date**, but there are important differences — and neither should ever be run *blindly* on a production system.

```
# Update all installed packages to the latest versions
sudo dnf update

# Upgrade all packages and handle obsolete ones
sudo dnf upgrade
```

Key Points:

- **`update`** → updates packages already installed.
- **`upgrade`** → updates packages *and* removes or replaces obsolete ones.
- Both can change critical libraries, services, or dependencies.

💡 **Reminder:**
These are not hot-run commands — always test in a **staging environment** first, review changelogs, and plan maintenance windows before applying full system updates.

Linux: Package Manager

🧊 Why We Use OS Images and Snapshots

Running `dnf update` or `dnf upgrade` in staging doesn't guarantee the same results in production — repositories change, and package versions can shift daily.

Instead of updating live systems, admins use **frozen OS images** or **snapshots**:

- **OS Images (AMIs, VM Templates):** Contain a *known good* system state with tested package versions.
- **Snapshots:** Capture a system's disk state before changes, allowing easy rollback.
- **Immutable Infrastructure:** Servers are rebuilt from images instead of being updated in place.

💡 **Key Idea:**
Staging is for *testing* — images are for *guaranteeing consistency.*
Use images to deploy identical, stable systems across environments.