Terminal Driver

❓ It’s worth noting that Linux preserves compatibility with physical terminal devices. The TTY Subsystem still handles both physical and virtual devices.

The Terminal Driver is part of the kernel, specifically part of the TTY Subsystem, which handles input and output for terminal devices.

It does not control the Terminal Emulator, but instead sits between user-space programs and terminal devices (shell, terminal emulator...).

The Terminal Driver:

- Manages Line Discipline (e.g. Canonical/Non-Canonical Modes)
- Handles special control characters (e.g. “Ctrl+C”)
- Controls input buffering, echoing, and signal generation
- Offers system calls like “tcsetattr()”, and “read()” to control or receive input/output from the terminal

When the user interacts with a terminal, the program actually interacts with the TTY Subsystem in the kernel, and not directly with the graphical Terminal Emulator.

What is the Terminal Emulator

A Terminal Emulator (e.g. “Konsole”, “GNOME Terminal”...) is a user-space GUI program that:

- Emulates a physical terminal
- Connects to a Pseudo-Terminal (PTY) device on the system
- Draws the screen, handles fonts/colors, and sends/receives bytes to/from the kernel’s TTY Subsystem

So it talks to the Terminal Driver via PTYs, just like a real hardware terminal would via a serial line.

Teletypewriter

> ❓ Teletypewriter is a legacy name that stuck around from early computing days.

A Teletypewriter (TTY) was a mechanical typewriter hooked up to a computer to type and read characters.

In UNIX, TTY became a shorthand for any (physical or virtual) terminal device that accepts input and provides output.

In Linux, TTYs are represented by device files like:

- “/dev/tty1” → A Virtual Console the user switches to with “Ctrl+Alt+F1”
- “/dev/ttyS0” → A Serial Port
- “/dev/tty” → The controlling terminal for the current process

In essence, a TTY is any terminal device recognized by the kernel that a program can use for I/O.

Pseudo-Terminal

> ❓ The “Y” in PTY can be misleading, because its meaning (Pseudo-Terminal) doesn’t include any “Y”.

A Pseudo-Terminal (PTY) is a software emulation of a physical terminal. Terminal Emulators like “xterm” or “konsole” use PTYs to simulate terminal behavior in a graphical environment.

PTYs come in pairs:

- One end is the Master “/dev/ptmx”
- The other end is the Slave “/dev/pts/N” where ‘N’ is the PTY number

This emulates a full-duplex communication channel, like a real serial cable connecting two devices.

Model of Master and Slave

This terminology is inherited from hardware terminology. It’s being phased out in modern documentation in favor of “controller/controlled”, between others.

- Master (Controller)
  - The Terminal Emulator process (e.g. “gnome-terminal”) connects here
  - Sends input to the Slave
  - Receives output from the Slave
- Slave (Controlled)
  - The shell (or user program) connects to this
  - The user program treats the Slave like a real TTY
  - Gets input from the Master
  - Writes output to the Master

Example:

When the user types in Konsole, its keystrokes go into the Master end.

They appear to the shell as coming from a real TTY (Slave end).

When the shell outputs text, it flows back through the Slave, to the Master, to Konsole. Finally, Konsole renders text on the screen.

> Terminal Emulator requests → Master → Slave → Shell

> Shell responds → Slave → Master → Terminal Emulator renders

Serial Port

A Serial Port is a hardware communication interface that sends data one bit at a time over a wire (serially), unlike parallel ports that send multiple bits simultaneously.

It’s an old but still-used interface (RS-232 standard).

On Linux, Serial Ports show up as:

- “/dev/ttyS0”, “/dev/ttyS1”...
- Serial Ports are controlled by the Serial driver in the Kernel

Serial Ports are still used today in some embedded systems or legacy devices. These represent real hardware devices attached to the computer.

If a physical terminal is connected to a Serial Port, the device is represented as a TTY device file (e.g. “/dev/ttyS0”). This is a real, physical terminal.

So the Serial Port and TTY are older concepts in Linux used for physical hardware communication, while the PTY model is a more modern software-based solution for simulating terminal behavior.

/dev/tty

The “/dev/tty” is a special device file in Unix-like systems that represents the Controlling Terminal of the current process.

The Controlling Terminal is typically associated with the process’s stdin, stdout, and stderr.

The Controlling Terminal is the terminal that was used to launch the process (e.g. a shell session or a Terminal Emulator).

For example, when a Terminal Emulator is opened (e.g. gnome-terminal), there’s a shell session running. There can be another program running within that shell (e.g. Vim).

The Controlling Terminal is the terminal where this process (shell or program) was started.

The purpose of “/dev/tty” is to give processes a way to interact with their Controlling Terminal, even if they aren’t directly reading from stdin or writing to stdout. In other words, it allows a process to refer to the terminal in an abstract way, no matter where its input/output is currently being directed.

- A process might lose access to its terminal if it’s run in the background. But if it wants to output something to the user or receive input, it can refer to “/dev/tty” and interact with its Controlling Terminal
- If the user runs a Terminal Emulator, the Controlling Terminal pointed to by “/dev/tty” is the Terminal Emulator itself
- If the user runs a process in a Virtual Console, the Controlling Terminal pointed to by “/dev/tty” is the Virtual Console

When a Terminal Emulator runs, opens the PTY Master, and forks a child:

- The shell is run in the child
- Using “dup2()”, the PTY Slave is set as the stdin/stdout/stderr of the child, which runs the shell
    - The child also opens “/dev/tty” which resolves to the Controlling Terminal: the PTY Slave

> User types a command (e.g. “ls -la”) and presses Enter. Terminal Emulator uses `write(masterFD, “ls -la\n”, 7)`

> Where “masterFD” represents PTY Master stdin data stream

> Data written to the PTY Master becomes available to be read from PTY Slave. In kernel space, the PTY pair is a bidirectional pipe, and the kernel bridges them

> Shell reads from its stdin `read(0)`, its stdin is pointing to the PTY Slave, so it reads the typed line “ls -la \n”

> Shell executes command and writes output to its stdout, which is the PTY Slave. “ls” subprocess writes output to FD 1 → PTY Slave

> Kernel sends the PTY Slave stdout to PTY Master stdout, which the Terminal emulator reads and displays

Virtual Console

A Virtual Console is a text-based terminal that allows users to interact with the system through a separate, full-screen session.

By default, the system can have several Virtual Consoles running simultaneously, each one providing a completely independent login and interactive shell session.

Virtual Consoles are managed by the kernel, and provide a way to have multiple independent text-based consoles, even if there's only a single physical terminal.

Virtual Consoles are typically accessible via function keys ("Ctrl+Alt+F1", "Ctrl+Alt+F2"...).

- Virtual Console 1 (/dev/tty1)
  - This is where the Display Manager and Graphical Environment runs
- Virtual Console 'n' (/dev/ttyn)
  - Use function keys to access tty2, tty3, tty4, tty5, and tty6
  - These consoles usually show a login prompt

A Virtual Console is a real kernel-managed terminal device backed by the Virtual Terminal Subsystem of the kernel (virtual terminal driver).

It's not a pseudo terminal, doesn't use Master-Slave model, and device files are "/dev/ttyN".

In the other hand, PTYs need a Terminal Emulator, and graphical output is rendered by the Display Server (e.g. X or Wayland).

The graphical output of a Virtual Console is a Text-Mode VGA Framebuffer or similar.

The Controlling Terminal of Virtual Console 2 would be “/dev/tty2”, the bash session would be started with stdin/stdout/stderr mapped to “/dev/tty2”, after running “/bin/login” to display the log in prompt.