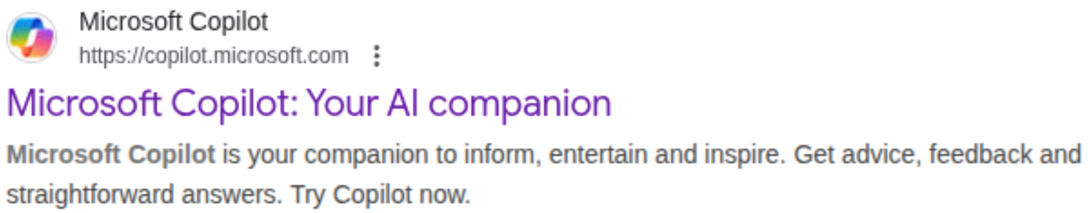
glibc Process Control

20251103 Mon

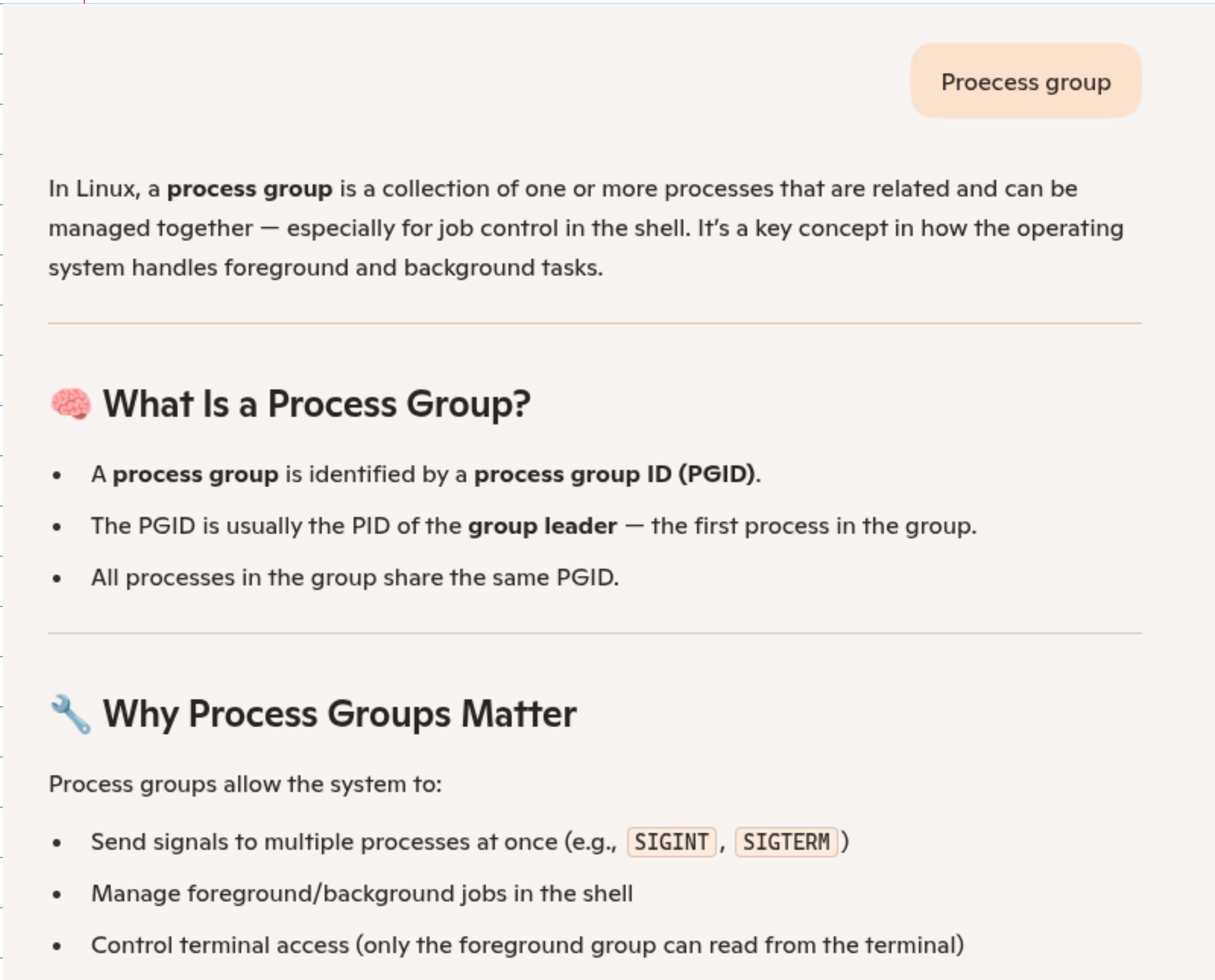
I used Microsoft Copilot to prepare this document



Microsoft Copilot
https://copilot.microsoft.com

Microsoft Copilot: Your AI companion

Microsoft Copilot is your companion to inform, entertain and inspire. Get advice, feedback and straightforward answers. Try Copilot now.



Proecess group

In Linux, a **process group** is a collection of one or more processes that are related and can be managed together — especially for job control in the shell. It's a key concept in how the operating system handles foreground and background tasks.

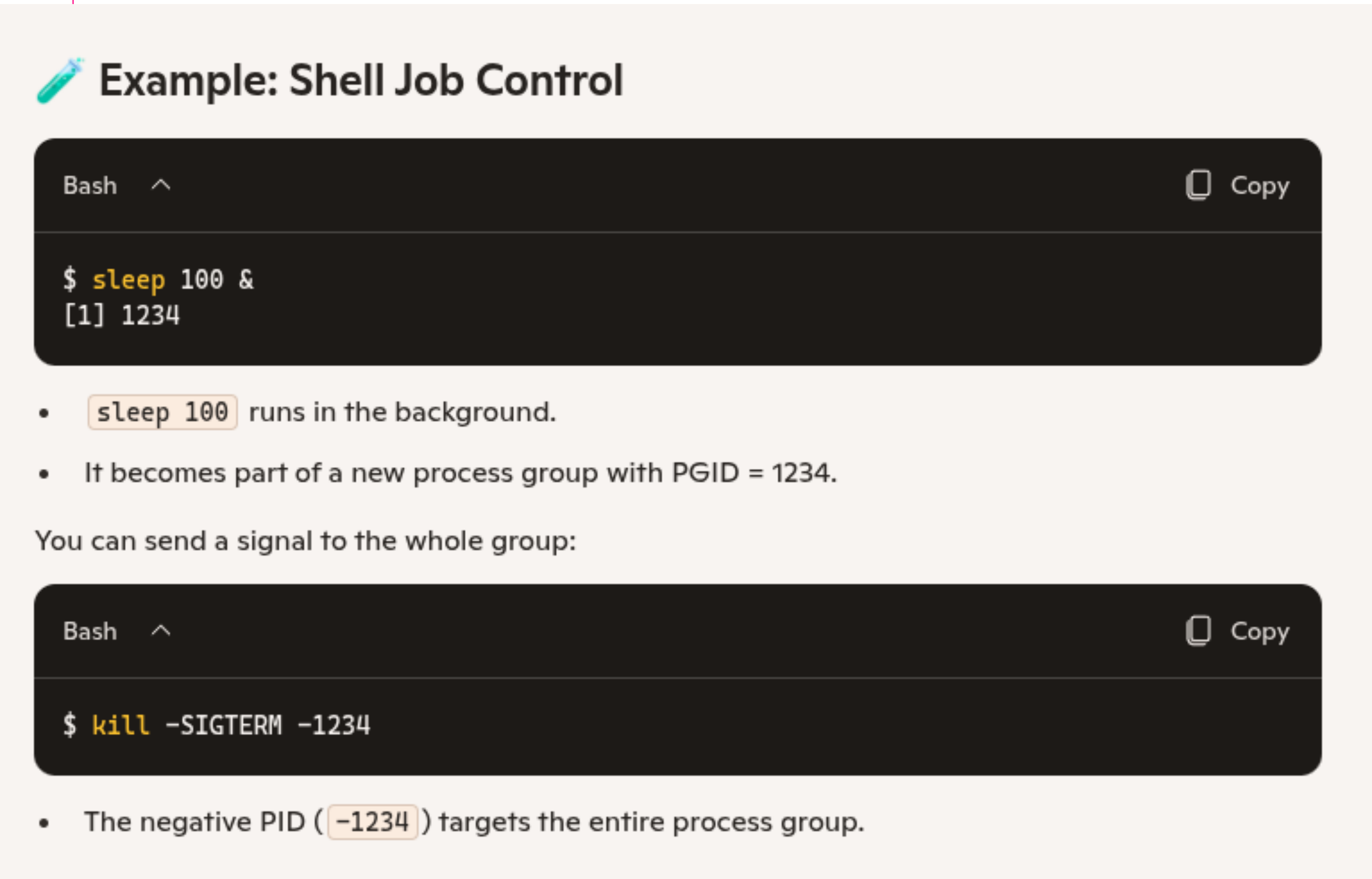
🧠 What Is a Process Group?

- A **process group** is identified by a **process group ID (PGID)**.
- The PGID is usually the PID of the **group leader** — the first process in the group.
- All processes in the group share the same PGID.

🔧 Why Process Groups Matter

Process groups allow the system to:

- Send signals to multiple processes at once (e.g., `SIGINT`, `SIGTERM`)
- Manage foreground/background jobs in the shell
- Control terminal access (only the foreground group can read from the terminal)



🧪 Example: Shell Job Control

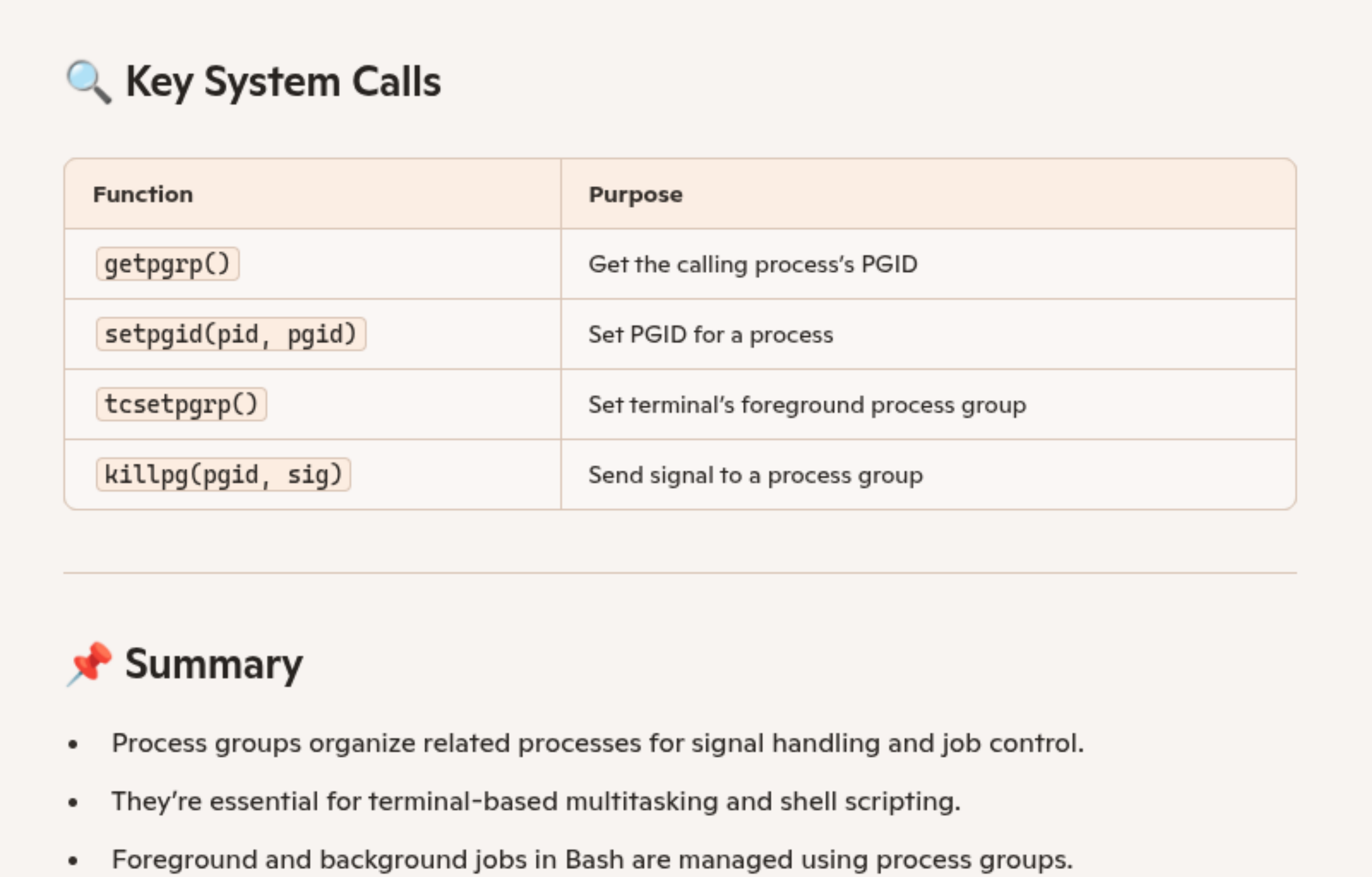
```bash
$ sleep 100 &
[1] 1234
```

- `sleep 100` runs in the background.
- It becomes part of a new process group with PGID = 1234.

You can send a signal to the whole group:

```bash
$ kill -SIGTERM -1234
```

- The negative PID (`-1234`) targets the entire process group.

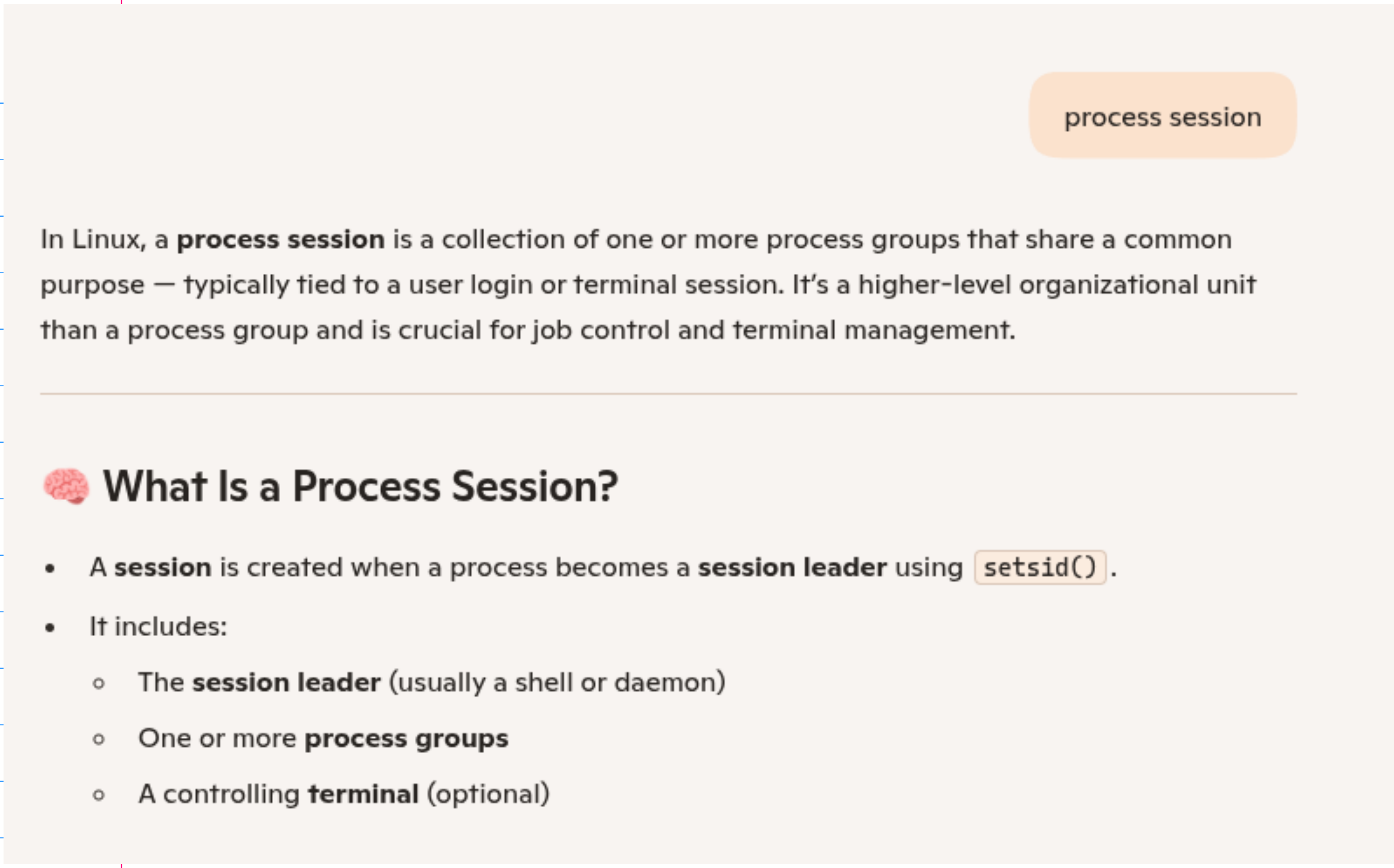


🔍 Key System Calls

Function	Purpose
`getpgrp()`	Get the calling process's PGID
`setpgid(pid, pgid)`	Set PGID for a process
`tcsetpgrp()`	Set terminal's foreground process group
`killpg(pgid, sig)`	Send signal to a process group

📌 Summary

- Process groups organize related processes for signal handling and job control.
- They're essential for terminal-based multitasking and shell scripting.
- Foreground and background jobs in Bash are managed using process groups.

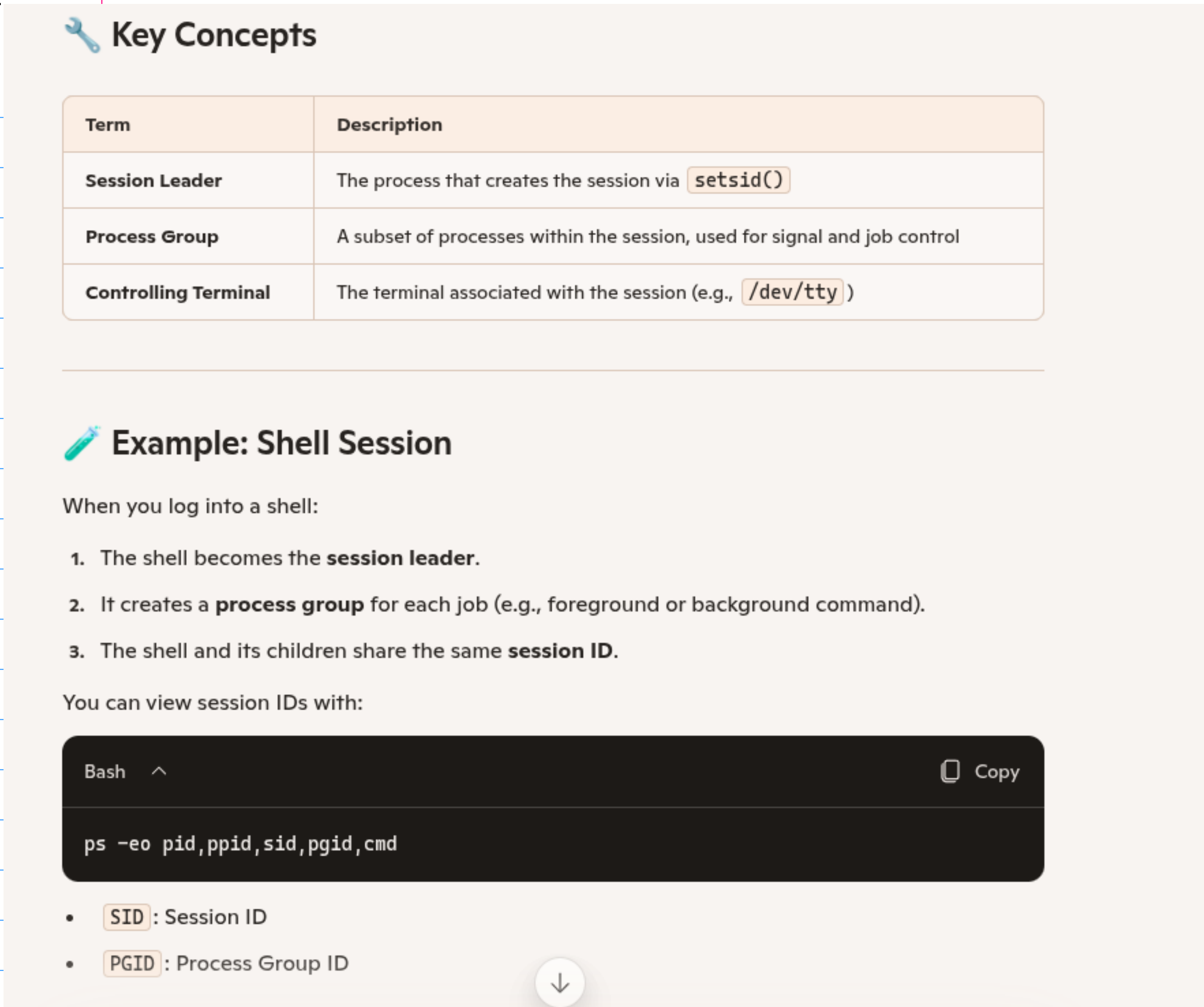


process session

In Linux, a **process session** is a collection of one or more process groups that share a common purpose — typically tied to a user login or terminal session. It's a higher-level organizational unit than a process group and is crucial for job control and terminal management.

🧠 What Is a Process Session?

- A **session** is created when a process becomes a **session leader** using `setsid()`.
- It includes:
  - The **session leader** (usually a shell or daemon)
  - One or more **process groups**
  - A controlling **terminal** (optional)



🔧 Key Concepts

Term	Description
Session Leader	The process that creates the session via `setsid()`
Process Group	A subset of processes within the session, used for signal and job control
Controlling Terminal	The terminal associated with the session (e.g., `/dev/tty`)

🧪 Example: Shell Session

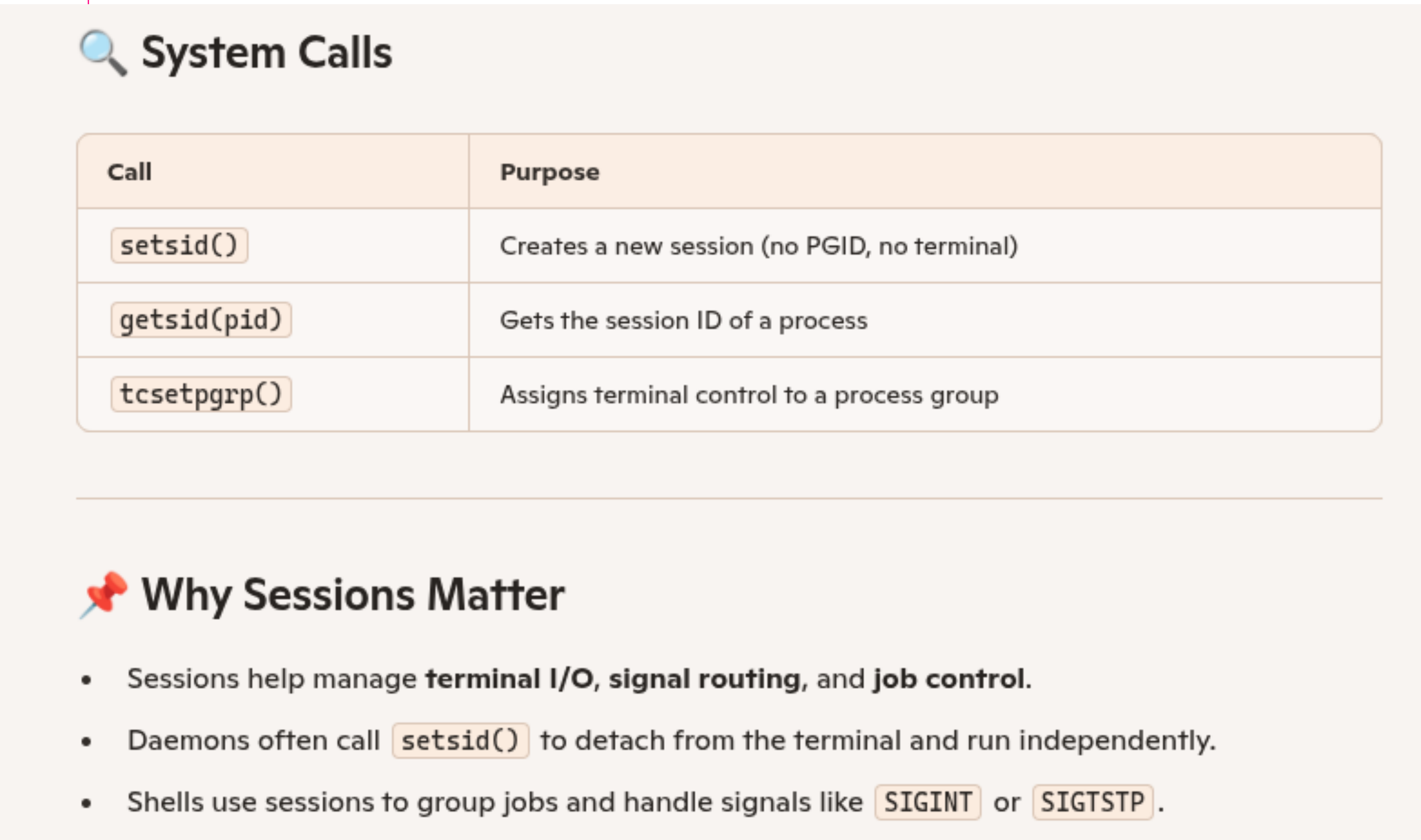
When you log into a shell:

1. The shell becomes the **session leader**.
2. It creates a **process group** for each job (e.g., foreground or background command).
3. The shell and its children share the same **session ID**.

You can view session IDs with:

```bash
ps -eo pid,ppid,sid,pgid,cmd
```

- `SID`: Session ID
- `PGID`: Process Group ID

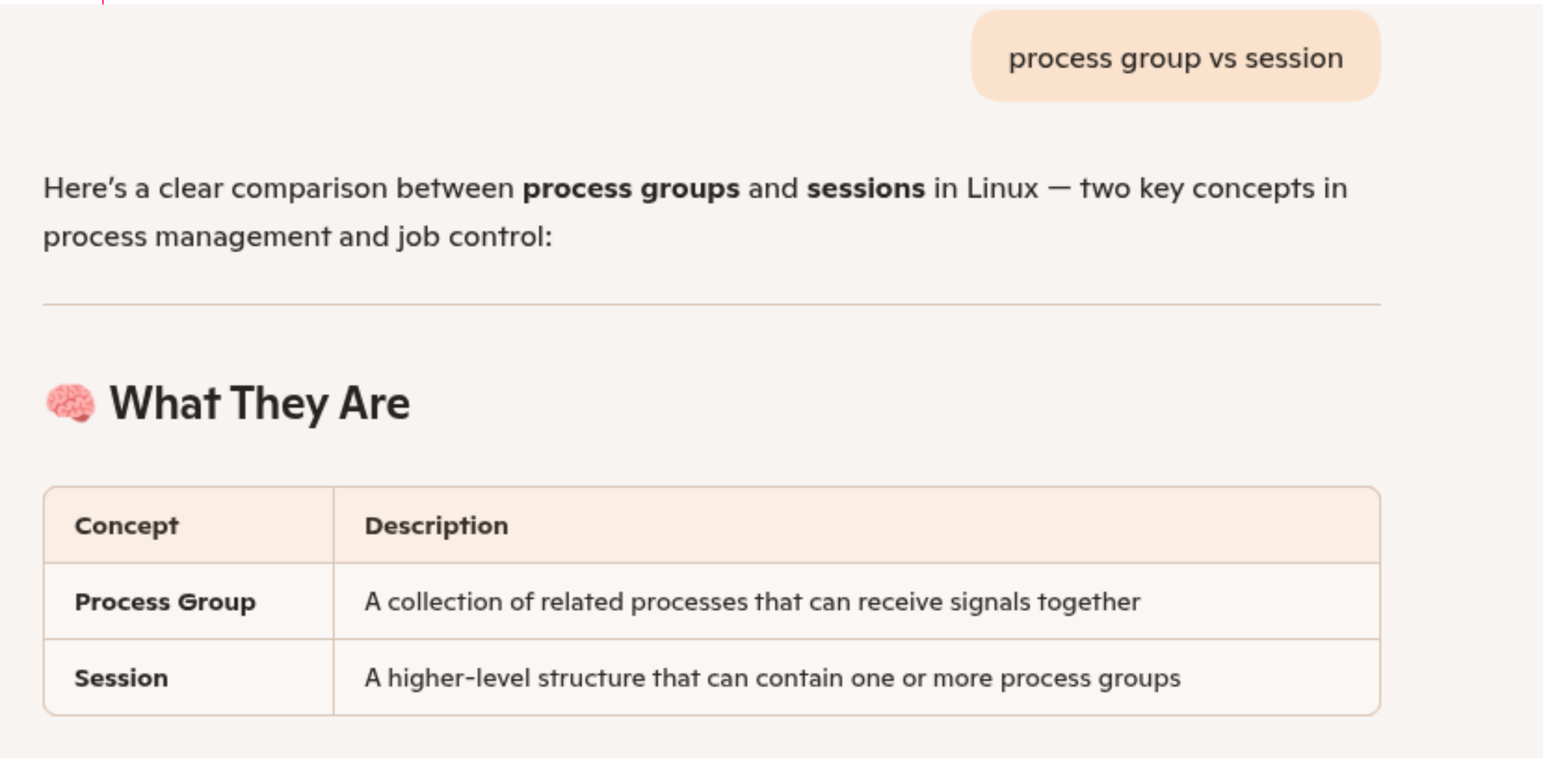


🔍 System Calls

Call	Purpose
`setsid()`	Creates a new session (no PGID, no terminal)
`getsid(pid)`	Gets the session ID of a process
`tcsetpgrp()`	Assigns terminal control to a process group

📌 Why Sessions Matter

- Sessions help manage **terminal I/O**, **signal routing**, and **job control**.
- Daemons often call `setsid()` to detach from the terminal and run independently.
- Shells use sessions to group jobs and handle signals like `SIGINT` or `SIGTSTP`.

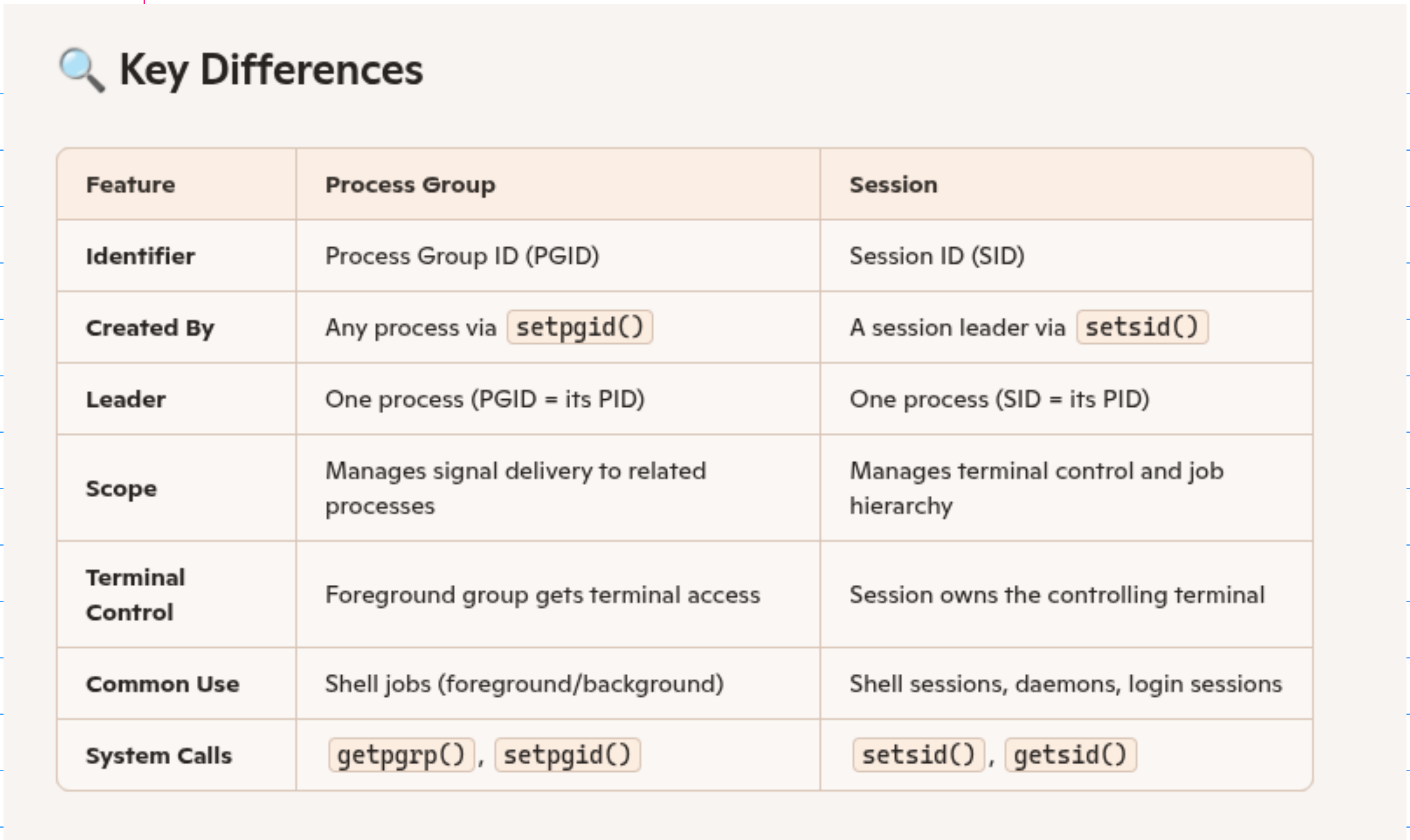


process group vs session

Here's a clear comparison between **process groups** and **sessions** in Linux — two key concepts in process management and job control:

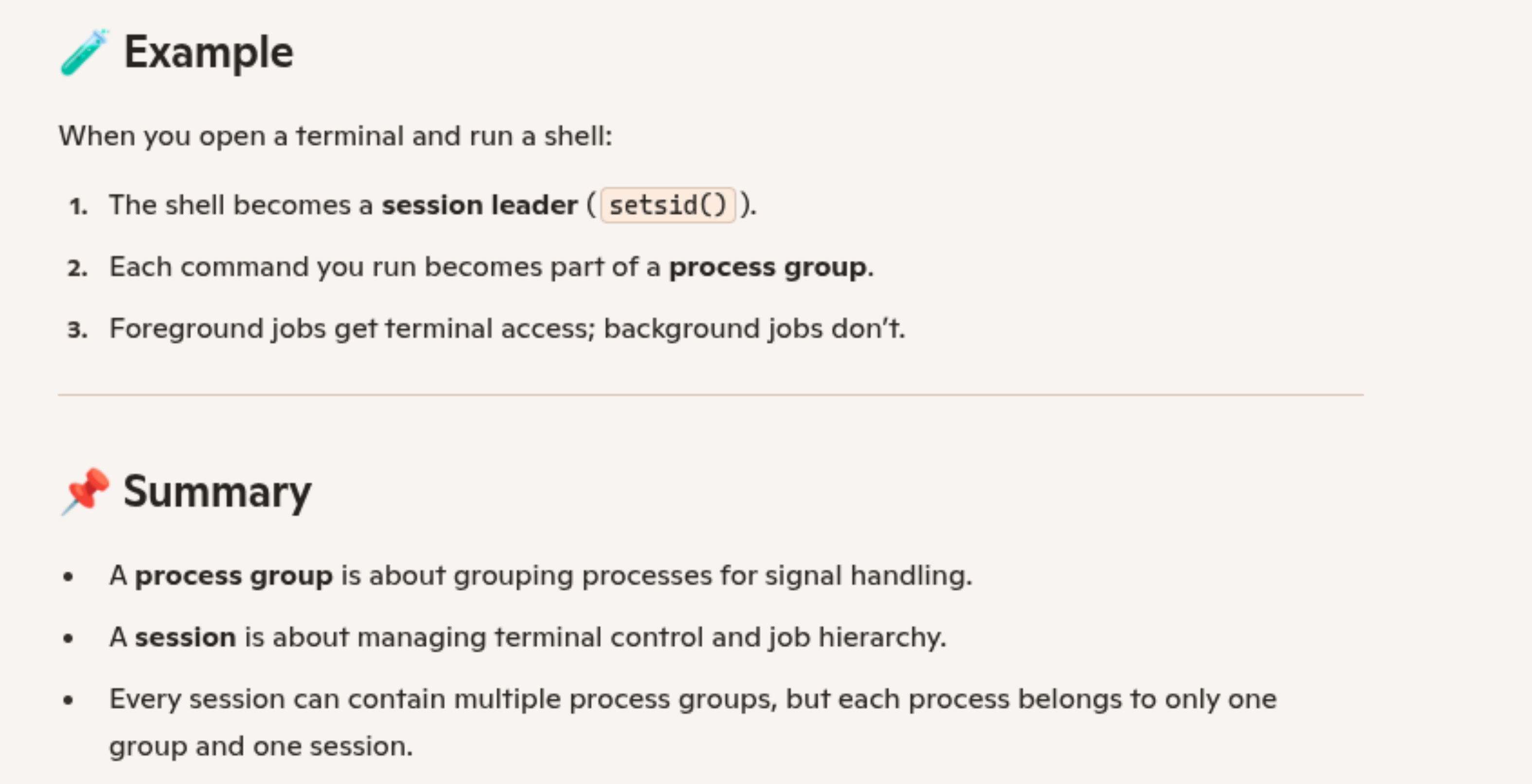
🧠 What They Are

Concept	**Description**
Process Group	A collection of related processes that can receive signals together
Session	A higher-level structure that can contain one or more process groups



🔍 Key Differences

Feature	Process Group	Session
Identifier	Process Group ID (PGID)	Session ID (SID)
Created By	Any process via `setpgid()`	A session leader via `setsid()`
Leader	One process (PGID = its PID)	One process (SID = its PID)
Scope	Manages signal delivery to related processes	Manages terminal control and job hierarchy
Terminal Control	Foreground group gets terminal access	Session owns the controlling terminal
Common Use	Shell jobs (foreground/background)	Shell sessions, daemons, login sessions
System Calls	`getpgrp()`, `setpgid()`	`setsid()`, `getsid()`



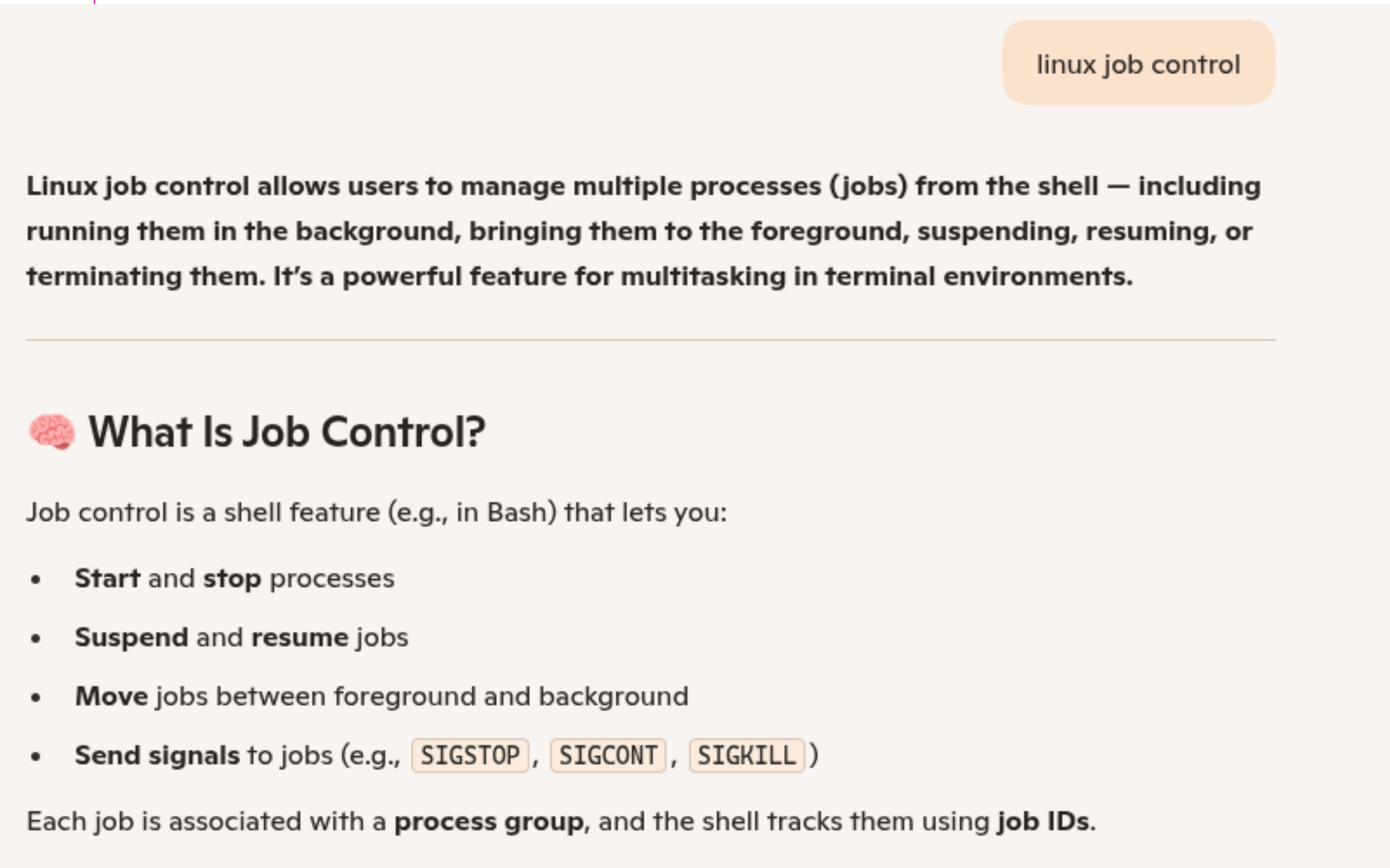
🧪 Example

When you open a terminal and run a shell:

1. The shell becomes a **session leader** (`setsid()`).
2. Each command you run becomes part of a **process group**.
3. Foreground jobs get terminal access; background jobs don't.

📌 Summary

- A **process group** is about grouping processes for signal handling.
- A **session** is about managing terminal control and job hierarchy.
- Every session can contain multiple process groups, but each process belongs to only one group and one session.



linux job control

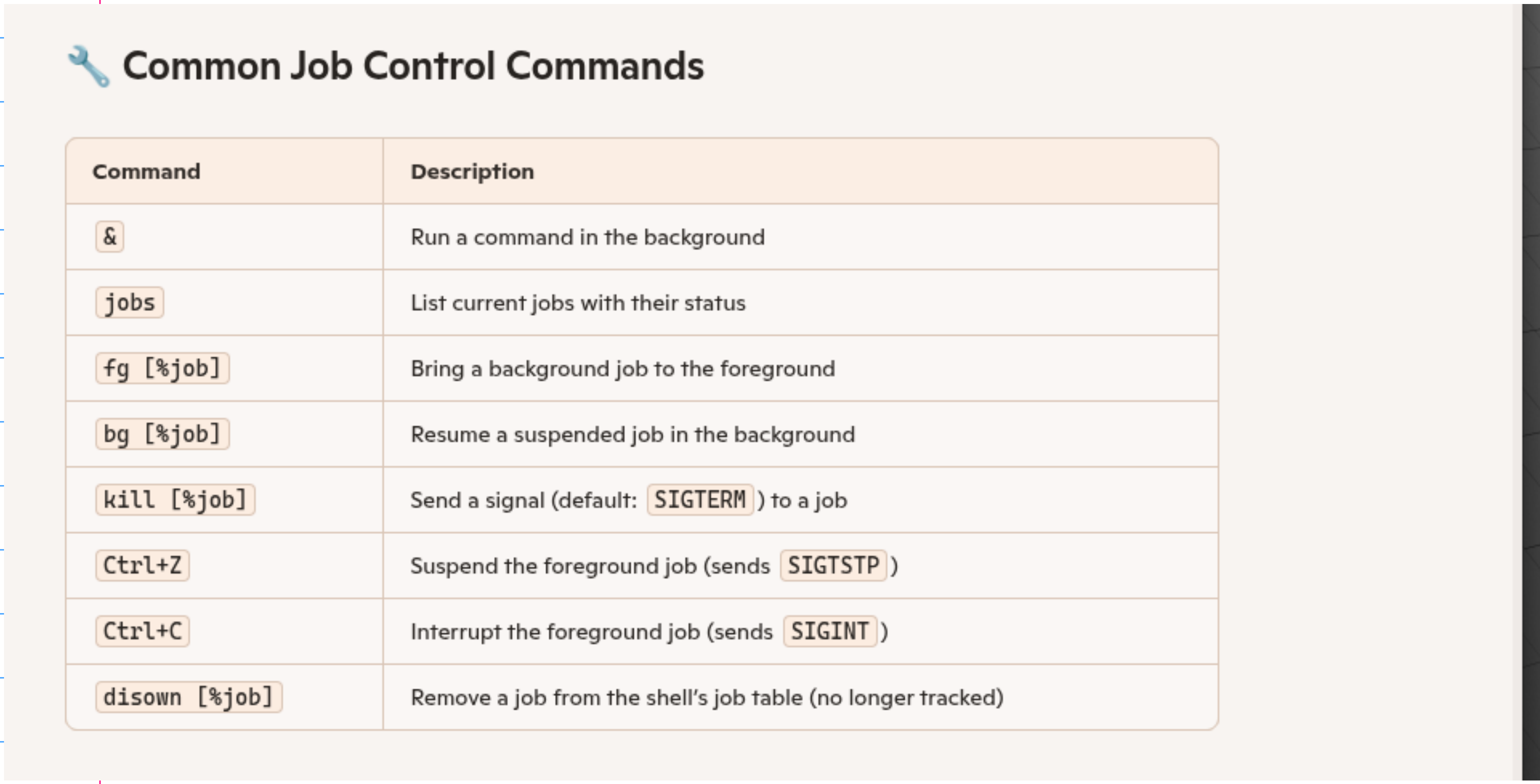
Linux job control allows users to manage multiple processes (jobs) from the shell — including running them in the background, bringing them to the foreground, suspending, resuming, or terminating them. It's a powerful feature for multitasking in terminal environments.

🧠 What Is Job Control?

Job control is a shell feature (e.g., in Bash) that lets you:

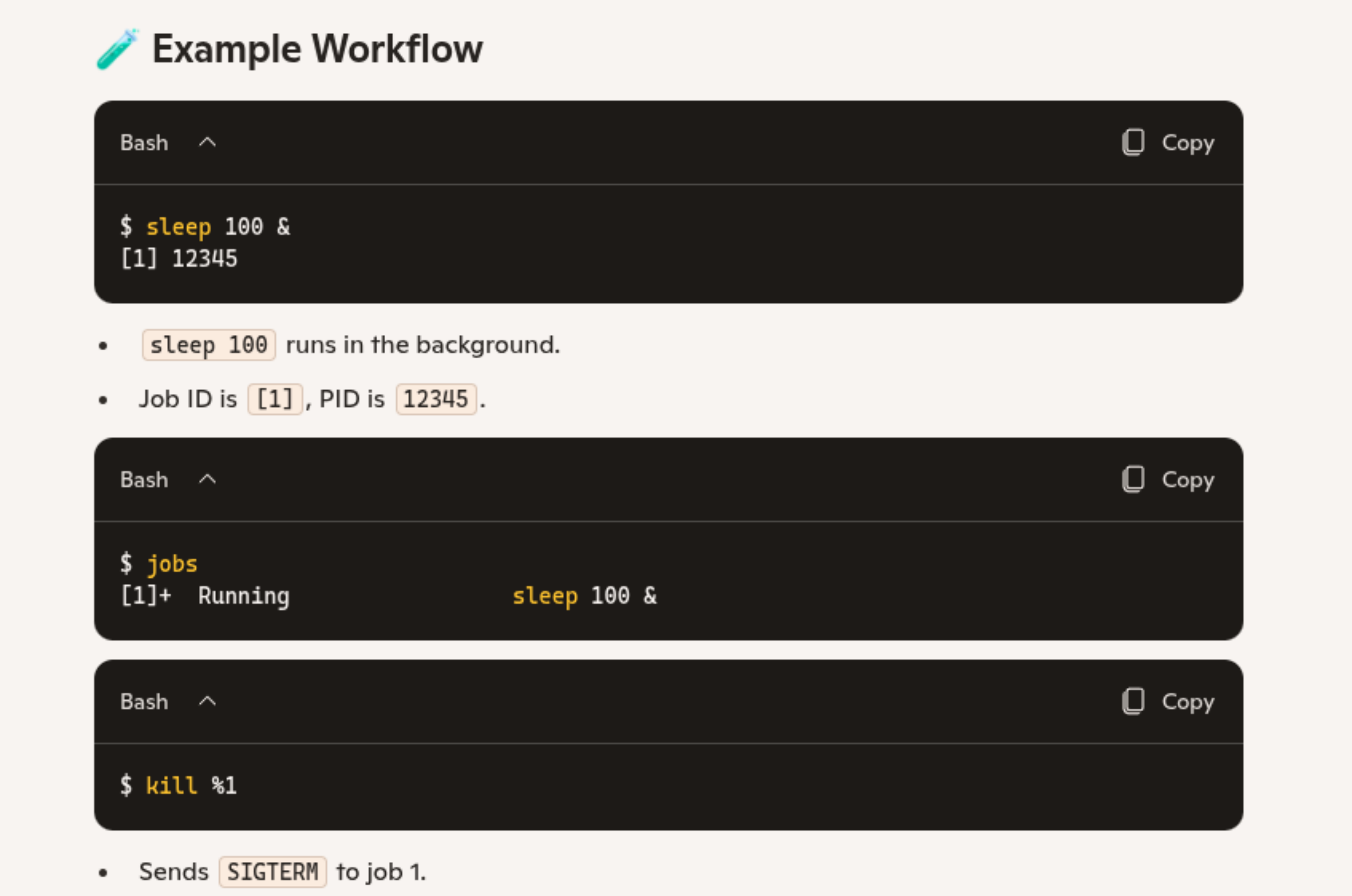
- **Start** and **stop** processes
- **Suspend** and **resume** jobs
- **Move** jobs between foreground and background
- **Send signals** to jobs (e.g., `SIGSTOP`, `SIGCONT`, `SIGKILL`)

Each job is associated with a **process group**, and the shell tracks them using **job IDs**.



🔧 Common Job Control Commands

Command	Description
`&`	Run a command in the background
`jobs`	List current jobs with their status
`fg [%job]`	Bring a background job to the foreground
`bg [%job]`	Resume a suspended job in the background
`kill [%job]`	Send a signal (default: `SIGTERM`) to a job
`Ctrl+Z`	Suspend the foreground job (sends `SIGTSTP`)
`Ctrl+C`	Interrupt the foreground job (sends `SIGINT`)
`disown [%job]`	Remove a job from the shell's job table (no longer tracked)



🧪 Example Workflow

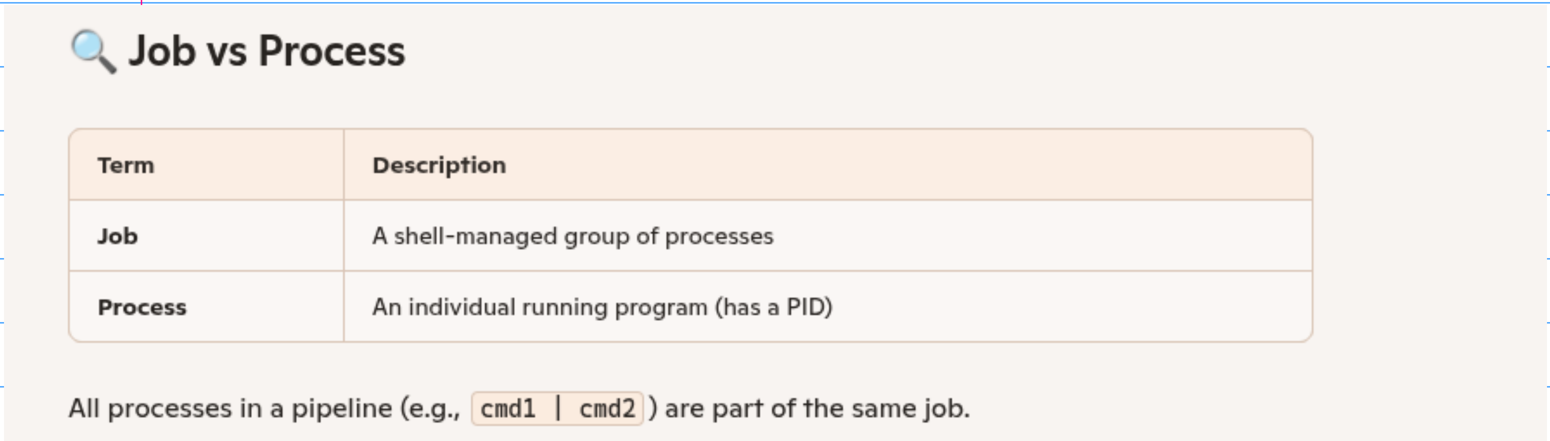
```bash
$ sleep 100 &
[1] 12345
```

- `sleep 100` runs in the background.
- Job ID is `[1]`, PID is `12345`.

```bash
$ jobs
[1]+  Running                 sleep 100 &
```

```bash
$ kill %1
```

- Sends `SIGTERM` to job 1.



🔍 Job vs Process

Term	Description
Job	A shell-managed group of processes
Process	An individual running program (has a PID)

All processes in a pipeline (e.g., `cmd1 | cmd2`) are part of the same job.

job control, process group, session

Here's a clear and cohesive explanation of how **job control, process groups**, and **sessions** work together in Linux. These concepts form the backbone of how the shell manages processes, especially in interactive environments.

🧠 1. Job Control

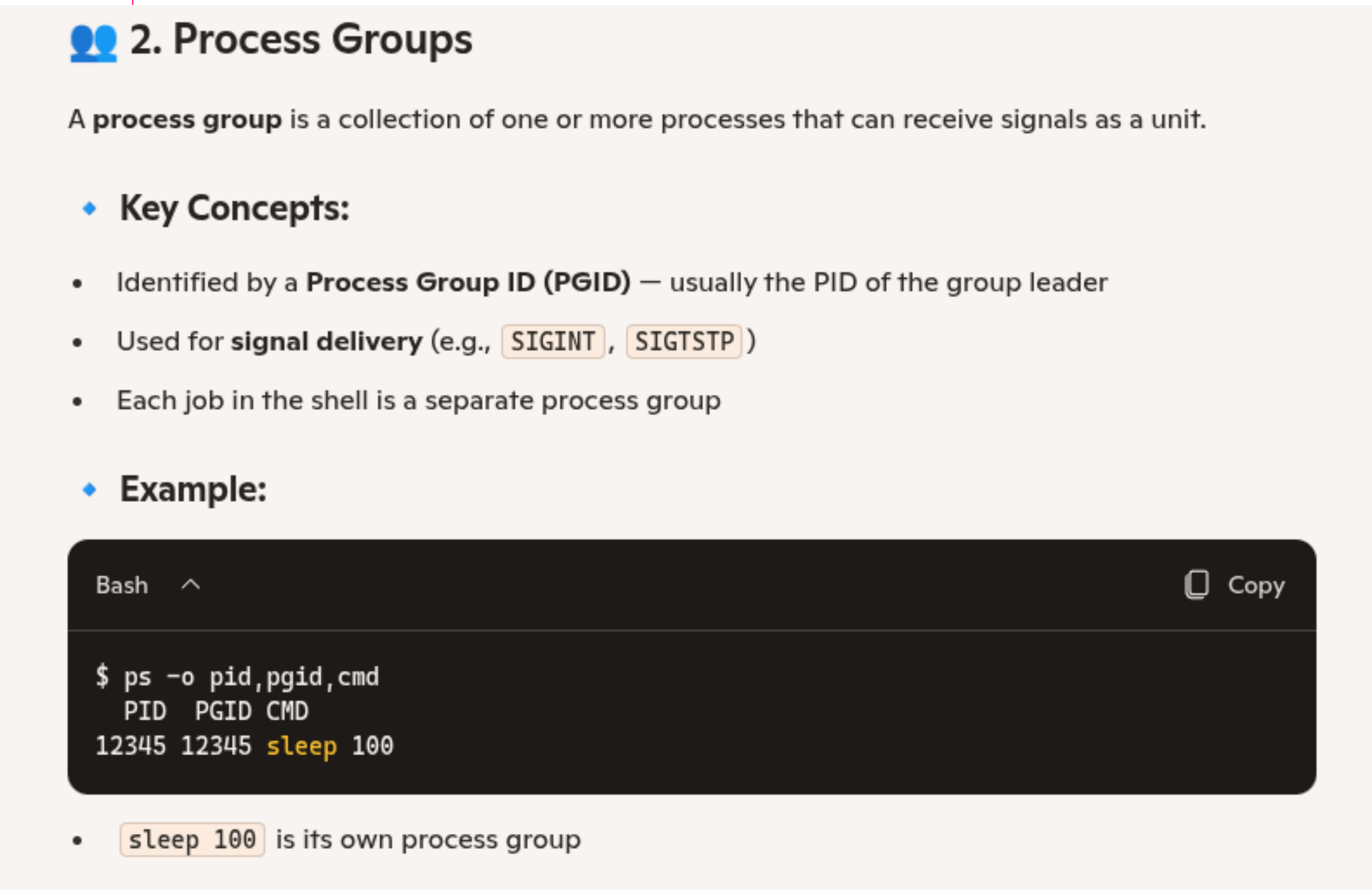
Job control is a shell feature that lets users manage multiple processes (jobs) from the command line.

🔹 Key Features:

- Run jobs in the **foreground** or **background**
- **Suspend** and **resume** jobs (`Ctrl+Z`, `bg`, `fg`)
- **List** jobs (`jobs`)
- **Send signals** to jobs (`kill`, `disown`)

Example:

```
sleep 100 &
] 12345
jobs
]+  Running                 sleep 100 &
kill %1
```



👥 2. Process Groups

A **process group** is a collection of one or more processes that can receive signals as a unit.

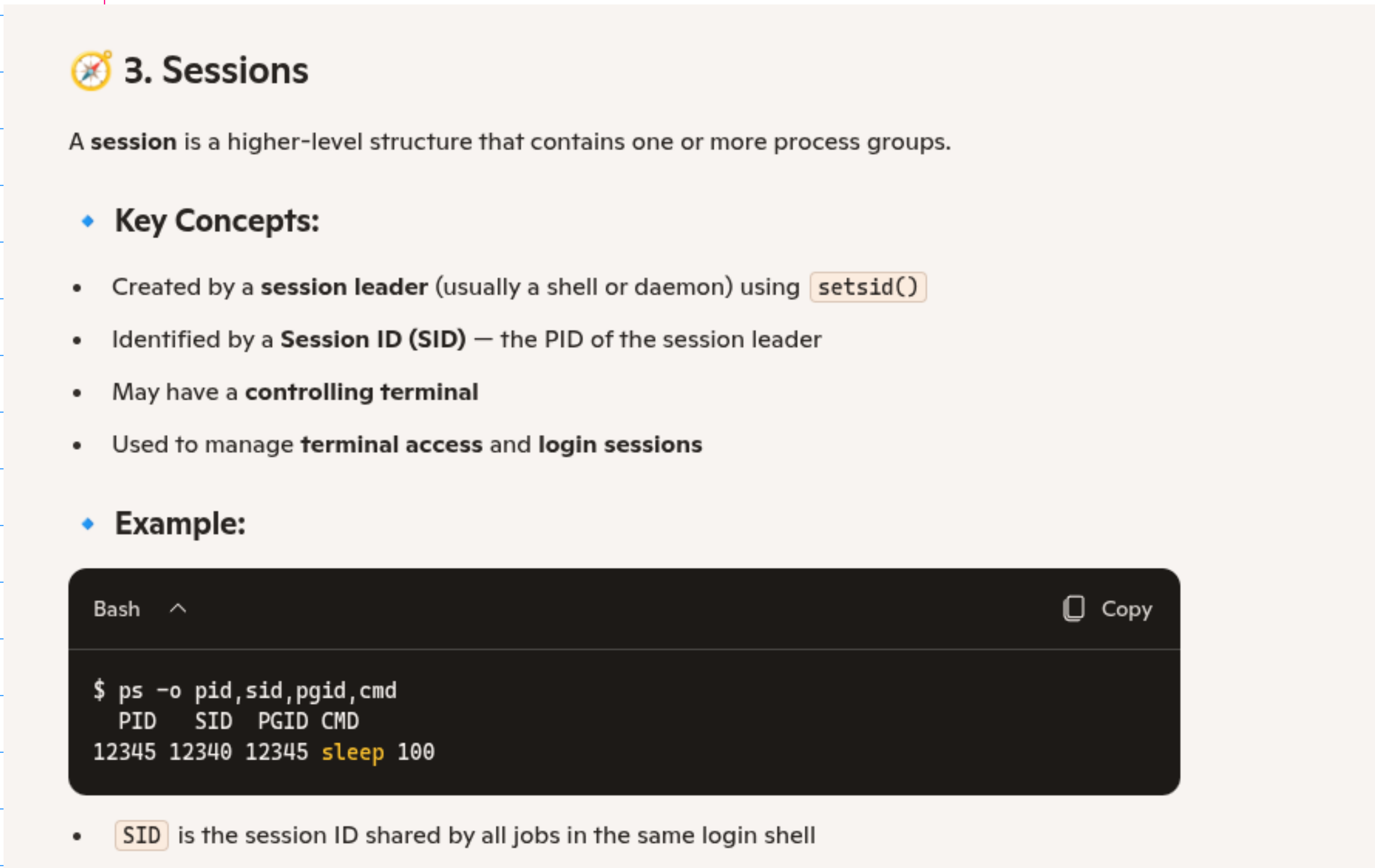
🔹 Key Concepts:

- Identified by a **Process Group ID (PGID)** — usually the PID of the group leader
- Used for **signal delivery** (e.g., `SIGINT`, `SIGTSTP`)
- Each job in the shell is a separate process group

🔹 Example:

```bash
$ ps -o pid,pgid,cmd
  PID  PGID CMD
12345 12345 sleep 100
```

- `sleep 100` is its own process group



🧭 3. Sessions

A **session** is a higher-level structure that contains one or more process groups.

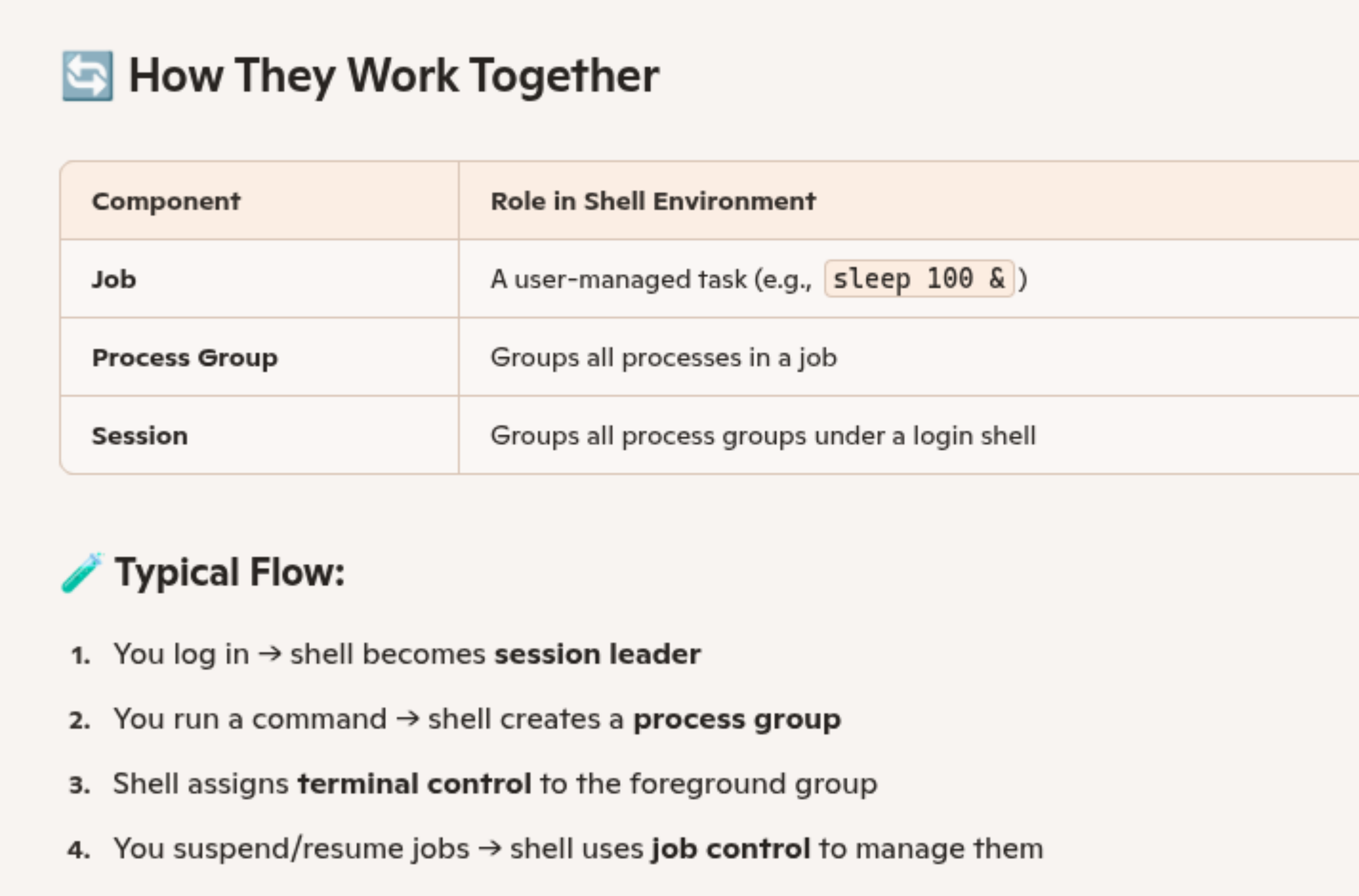
🔹 Key Concepts:

- Created by a **session leader** (usually a shell or daemon) using `setsid()`
- Identified by a **Session ID (SID)** — the PID of the session leader
- May have a **controlling terminal**
- Used to manage **terminal access** and **login sessions**

🔹 Example:

```bash
$ ps -o pid,sid,pgid,cmd
  PID   SID  PGID CMD
12345 12340 12345 sleep 100
```

- `SID` is the session ID shared by all jobs in the same login shell

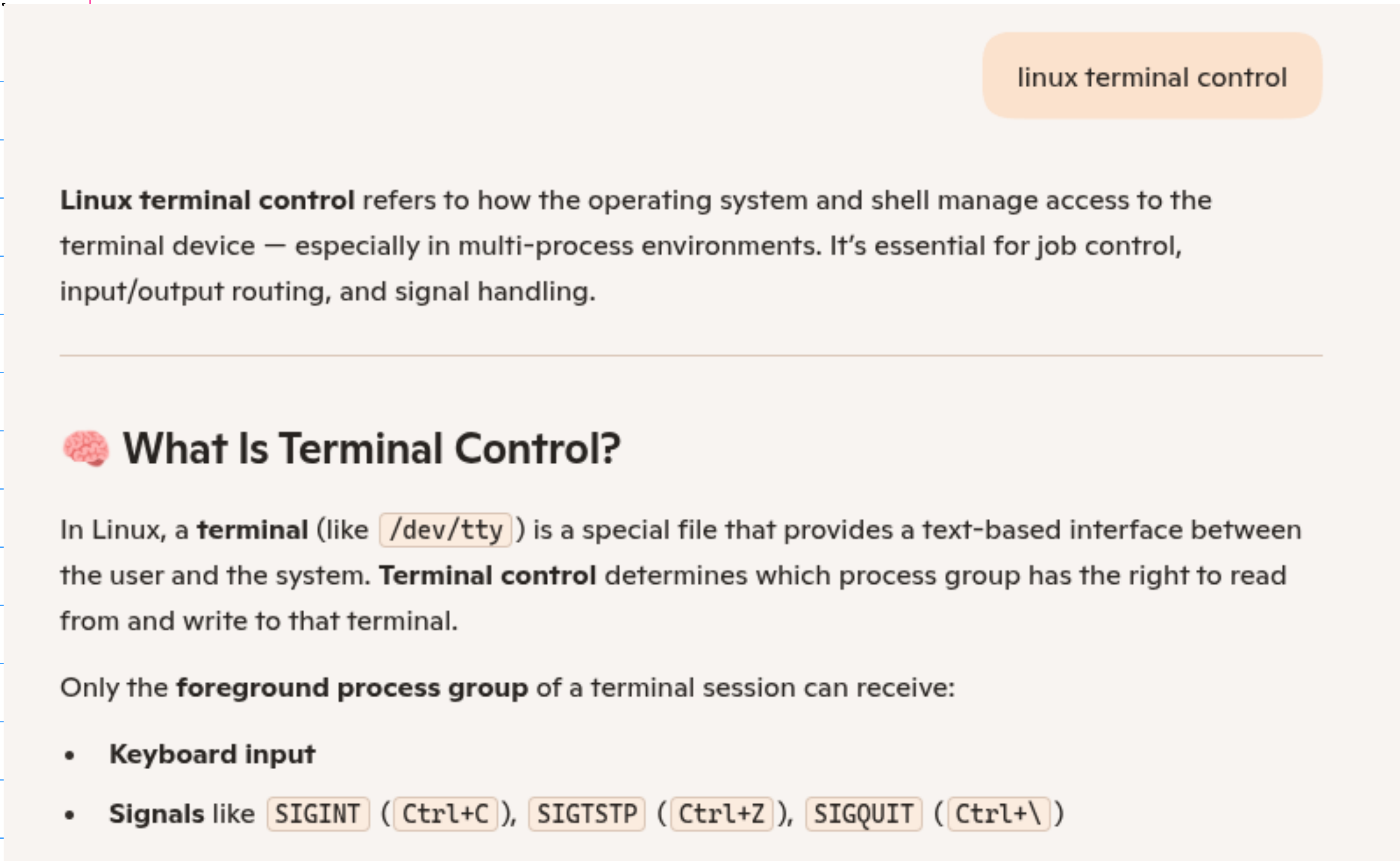


🔄 How They Work Together

Component	Role in Shell Environment
Job	A user-managed task (e.g., `sleep 100 &`)
Process Group	Groups all processes in a job
Session	Groups all process groups under a login shell

🧪 Typical Flow:

1. You log in → shell becomes **session leader**
2. You run a command → shell creates a **process group**
3. Shell assigns **terminal control** to the foreground group
4. You suspend/resume jobs → shell uses **job control** to manage them



linux terminal control

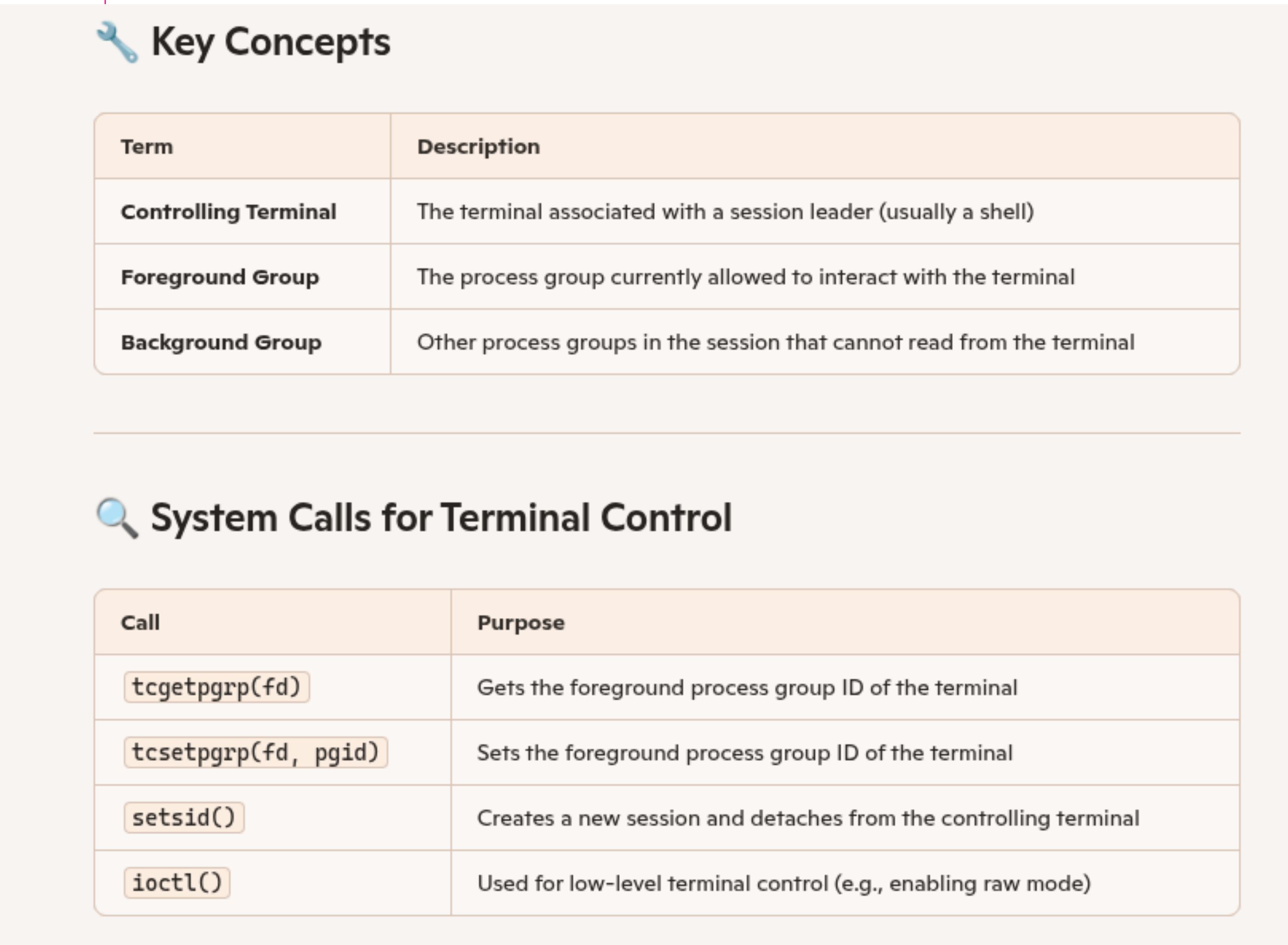
Linux terminal control refers to how the operating system and shell manage access to the terminal device — especially in multi-process environments. It's essential for job control, input/output routing, and signal handling.

🧠 What Is Terminal Control?

In Linux, a **terminal** (like `/dev/tty`) is a special file that provides a text-based interface between the user and the system. **Terminal control** determines which process group has the right to read from and write to that terminal.

Only the **foreground process group** of a terminal session can receive:

- **Keyboard input**
- **Signals** like `SIGINT` (`Ctrl+C`), `SIGTSTP` (`Ctrl+Z`), `SIGQUIT` (`Ctrl+\`)



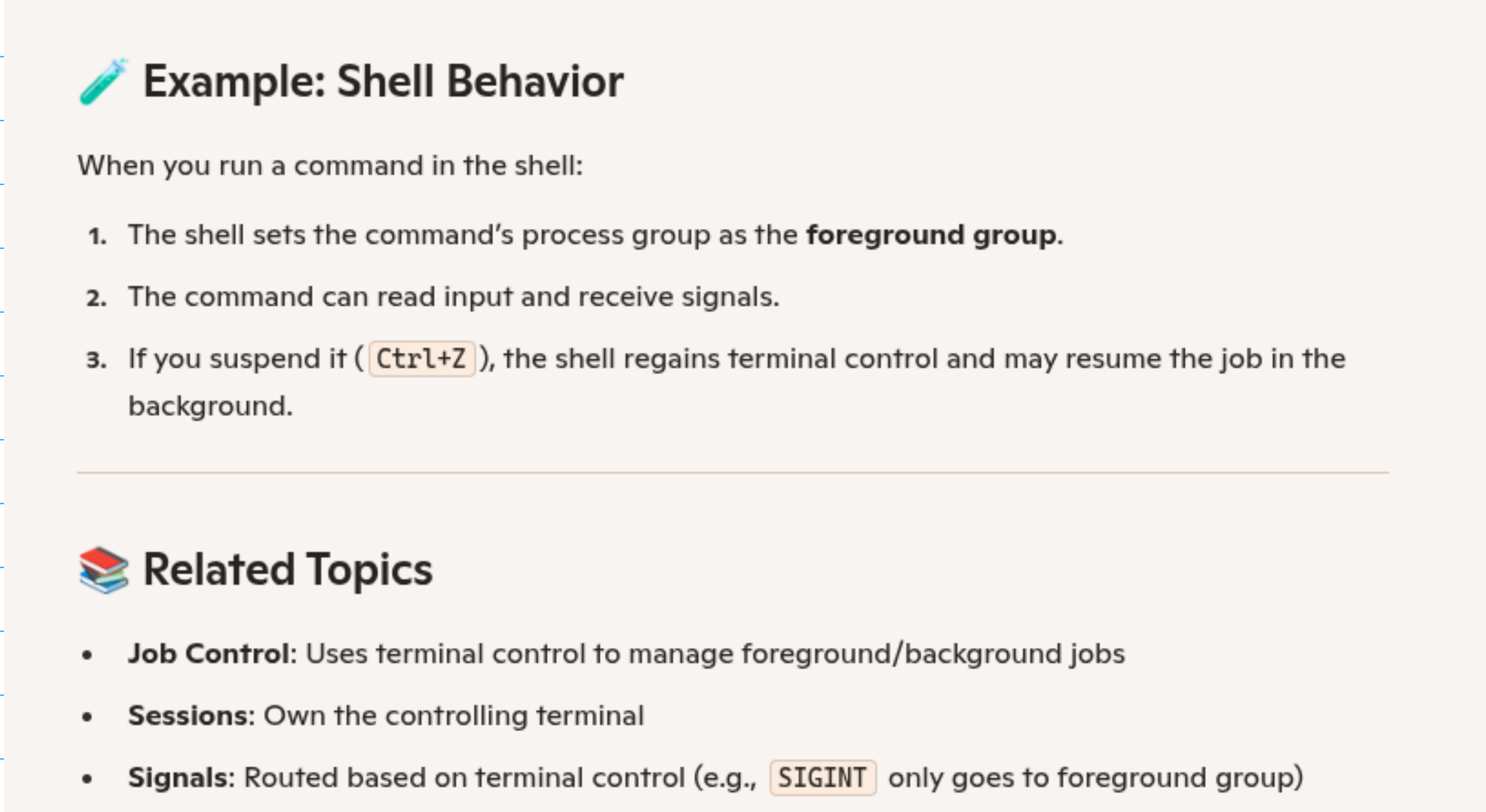
🔧 Key Concepts

Term	Description
Controlling Terminal	The terminal associated with a session leader (usually a shell)
Foreground Group	The process group currently allowed to interact with the terminal
Background Group	Other process groups in the session that cannot read from the terminal

---

🔍 System Calls for Terminal Control

Call	Purpose
`tcgetpgrp(fd)`	Gets the foreground process group ID of the terminal
`tcsetpgrp(fd, pgid)`	Sets the foreground process group ID of the terminal
`setsid()`	Creates a new session and detaches from the controlling terminal
`ioctl()`	Used for low-level terminal control (e.g., enabling raw mode)



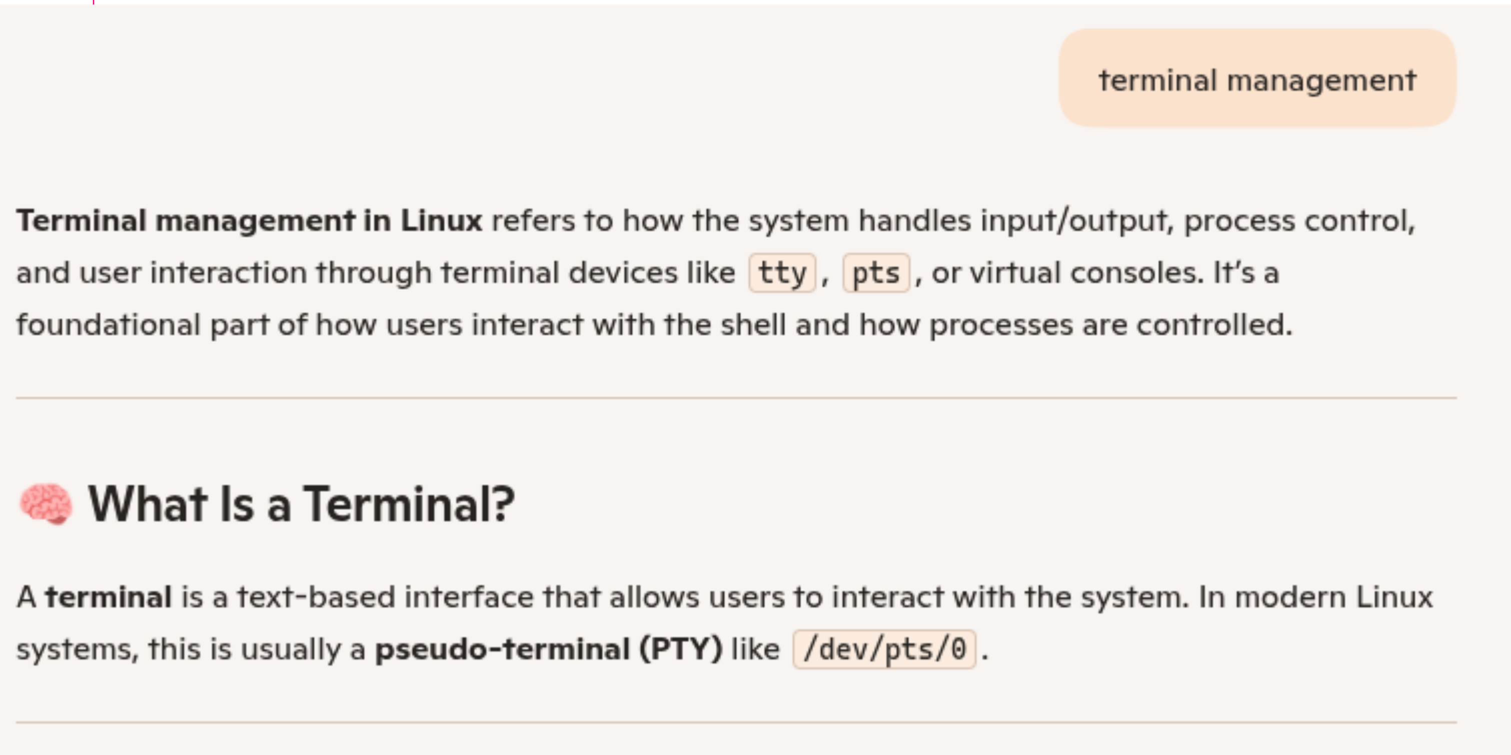
🧪 Example: Shell Behavior

When you run a command in the shell:

1. The shell sets the command's process group as the **foreground group**.
2. The command can read input and receive signals.
3. If you suspend it (`Ctrl+Z`), the shell regains terminal control and may resume the job in the background.

📚 Related Topics

- **Job Control**: Uses terminal control to manage foreground/background jobs
- **Sessions**: Own the controlling terminal
- **Signals**: Routed based on terminal control (e.g., `SIGINT` only goes to foreground group)

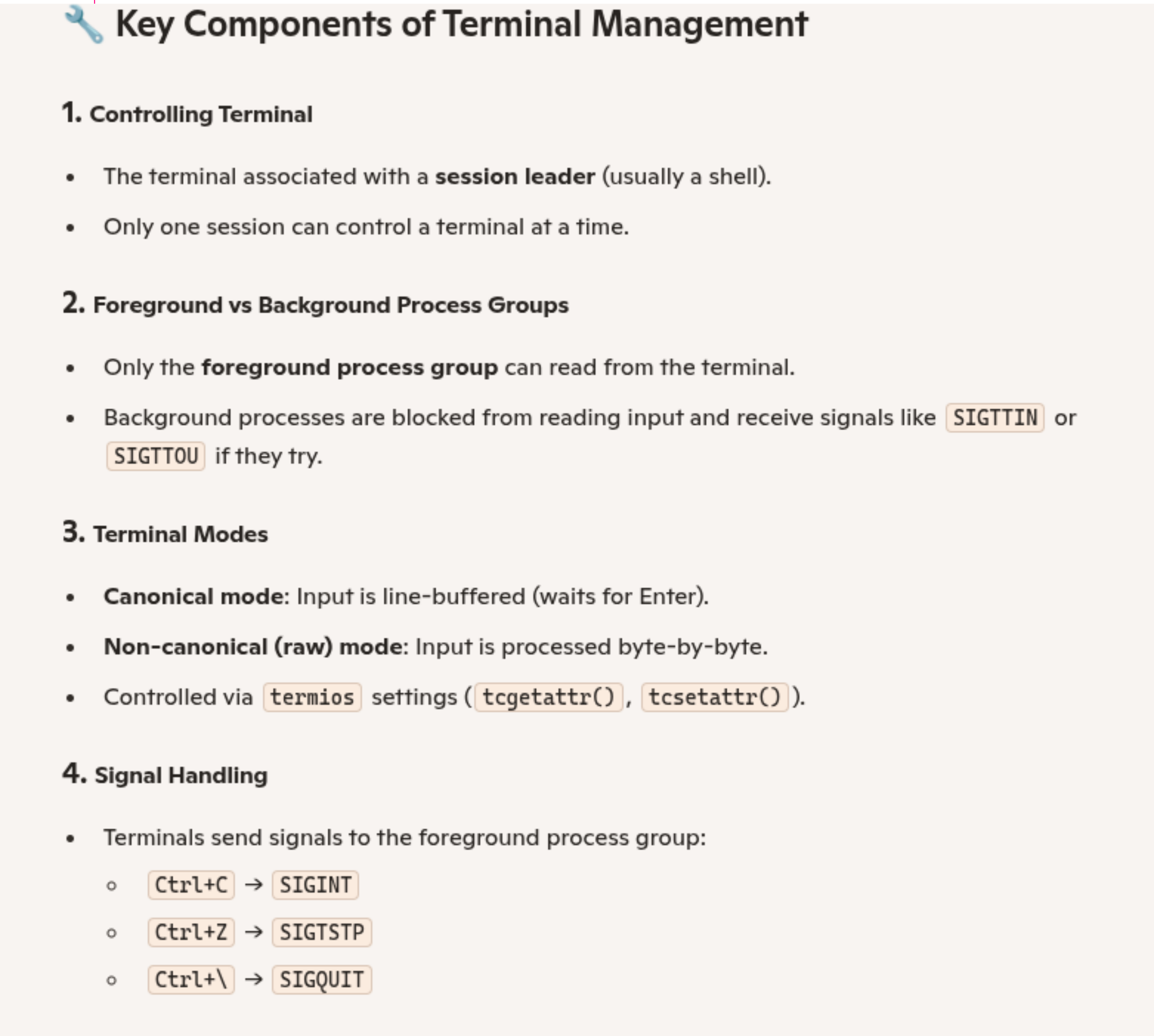


terminal management

Terminal management in Linux refers to how the system handles input/output, process control, and user interaction through terminal devices like `tty`, `pts`, or virtual consoles. It's a foundational part of how users interact with the shell and how processes are controlled.

🧠 What Is a Terminal?

A **terminal** is a text-based interface that allows users to interact with the system. In modern Linux systems, this is usually a **pseudo-terminal (PTY)** like `/dev/pts/0`.



🔧 Key Components of Terminal Management

1. Controlling Terminal

- The terminal associated with a **session leader** (usually a shell).
- Only one session can control a terminal at a time.

2. Foreground vs Background Process Groups

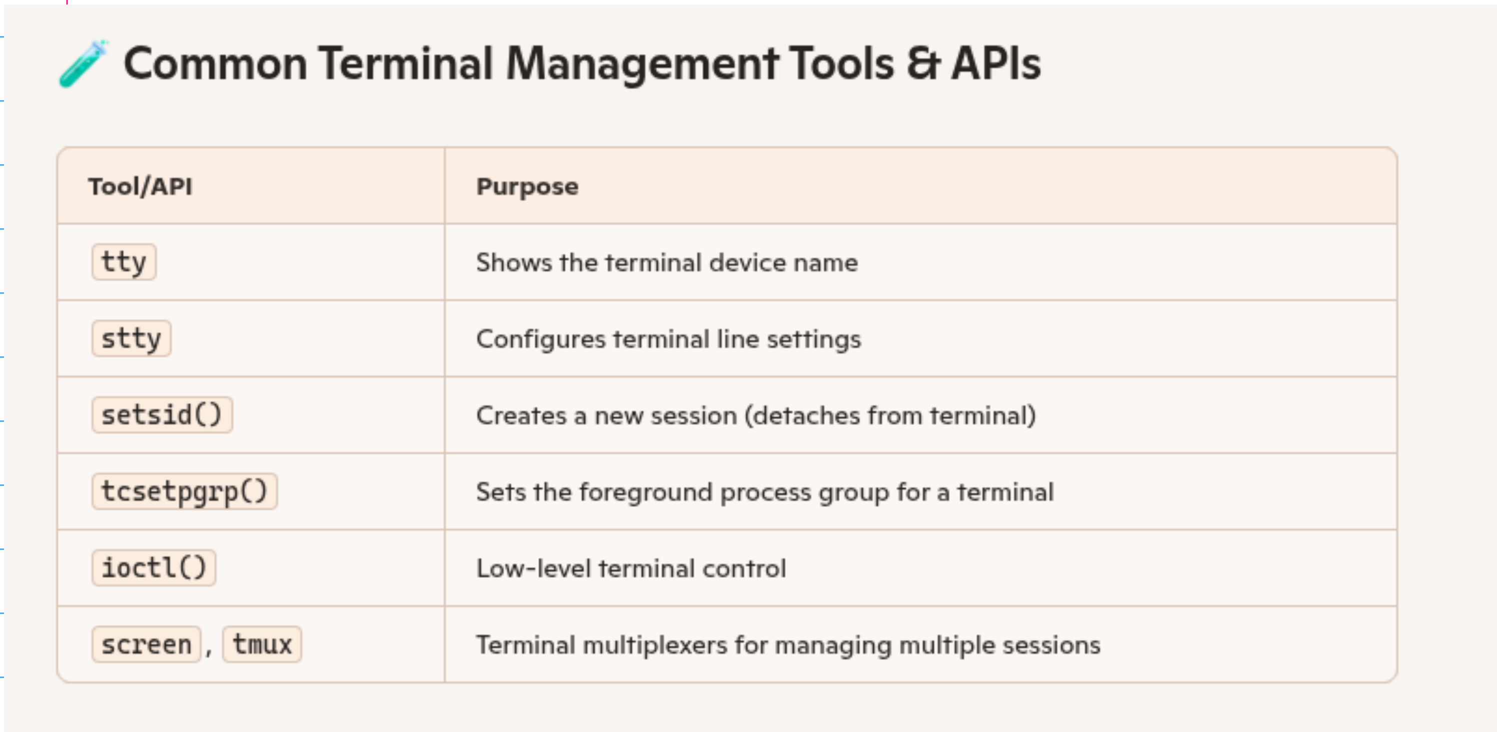
- Only the **foreground process group** can read from the terminal.
- Background processes are blocked from reading input and receive signals like `SIGTTIN` or `SIGTTOU` if they try.

3. Terminal Modes

- **Canonical mode**: Input is line-buffered (waits for Enter).
- **Non-canonical (raw) mode**: Input is processed byte-by-byte.
- Controlled via `termios` settings (`tcgetattr()`, `tcsetattr()`).

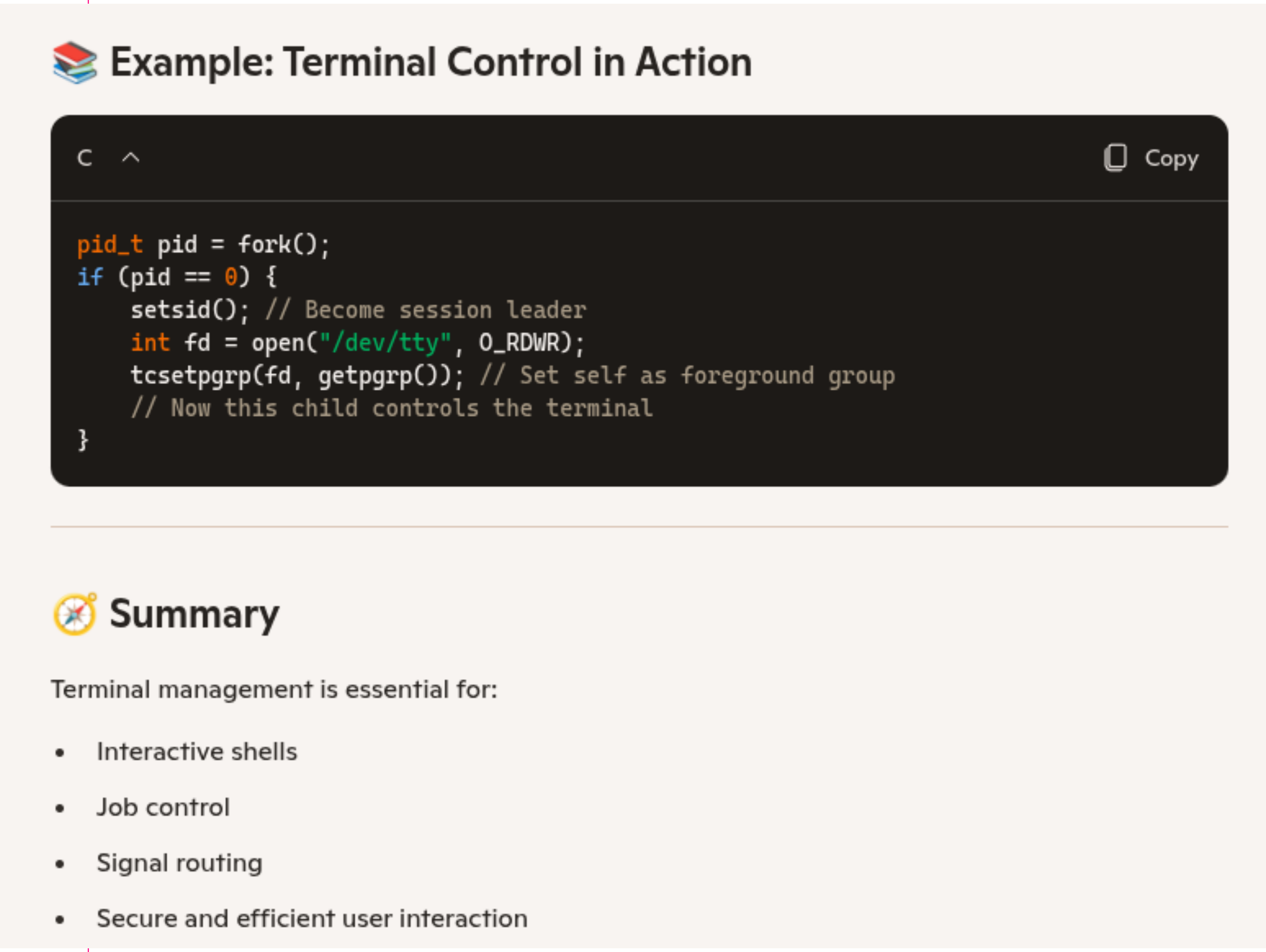
4. Signal Handling

- Terminals send signals to the foreground process group:
    - `Ctrl+C` → `SIGINT`
    - `Ctrl+Z` → `SIGTSTP`
    - `Ctrl+\` → `SIGQUIT`



🧪 Common Terminal Management Tools & APIs

Tool/API	Purpose
`tty`	Shows the terminal device name
`stty`	Configures terminal line settings
`setsid()`	Creates a new session (detaches from terminal)
`tcsetpgrp()`	Sets the foreground process group for a terminal
`ioctl()`	Low-level terminal control
`screen`, `tmux`	Terminal multiplexers for managing multiple sessions



📚 Example: Terminal Control in Action

```c
pid_t pid = fork();
if (pid == 0) {
    setsid(); // Become session leader
    int fd = open("/dev/tty", O_RDWR);
    tcsetpgrp(fd, getpgrp()); // Set self as foreground group
    // Now this child controls the terminal
}
```

🧭 Summary

Terminal management is essential for:

- Interactive shells
- Job control
- Signal routing
- Secure and efficient user interaction