That's an excellent way to structure high-level learning: **Fragmented Discovery** tied to **Boolean Server Logic** for controlled, challenging progression. This ensures players master individual concepts before combining them into advanced scripts.
Here is the design for the multi-step learn-to-unlock and command usage system.

Fragmented Knowledge and Boolean Logic Progression 🧩

The core idea is that no single file or action unlocks an advanced command. The player must collect and synthesize multiple **Knowledge Fragments** to form a complete understanding. The server's logic gates their progression using simple **Boolean (TRUE/FALSE)** checks on the player's knowledge state.

1. Multi-Step Learn-to-Unlock Process

Advanced commands (like thread spawn or raw send) are locked behind a three-stage progression: **Fragment Collection**, **Synthesis**, and **Validation**.

Stage	Action Required by Player	Resulting Knowledge State
Stage 1: Fragment Collection	Player must find **3 unique Knowledge Fragments** relating to the command (e.g., a header file, a configuration snippet, and a log entry).	Player has necessary resources, but the command is **not executable**.
Stage 2: Synthesis	Player must use the **NexusScript Editor** (edit) to compile the fragments into a single, syntactically correct **Knowledge Module** (.kns).	The command's man page becomes viewable (man thread spawn), but the command is **non-functional**.
Stage 3: Validation	Player must successfully execute a mission or a high-level command that requires the *concept* (not the full command) to prove mastery.	The command is **unlocked and fully functional**.

2. Boolean Server Logic Gates (The Control Mechanism)

The game server uses simple binary checks to manage the player's progress. This high-level logic uses minimal resources but maintains a complex gate system.

Example: Unlocking the thread spawn command (Parallel Processing)

The server checks three key Boolean states on the player's profile:

State Variable (Boolean)	Set to TRUE by Player Action (The GOAL)	Player Logic Learned
KNOWLEDGE_FRAGMENTS_COLLECTED	**GOAL:** Find and copy 3 files (.cfg, .h, .log) from high-level	**Logic:** File System I/O and Reconnaissance.

State Variable (Boolean)	Set to TRUE by Player Action (The GOAL)	Player Logic Learned
	targets (PvE or PvP).	
SYNTHESIS_MODULE_CREATED	**GOAL:** Write the thread_spawn.kns file that correctly references the 3 fragments.	**Logic:** NexusScript Syntax and Module Structuring.
CONCEPT_VALIDATED	**GOAL:** Complete a specific high-level mission that requires *simultaneous actions* (e.g., launching an attack while running a defense script), even without the full thread spawn command.	**Logic:** Foundational understanding of concurrency and resource management.

Unlock Gate Logic:

The server only enables the **full functionality** of the thread spawn command when all conditions are met:
If the logic gate evaluates to **TRUE**, the player's VC kernel integrates the command.

3. Usage and Code Experience Gain

Once a command is unlocked, its use drives the player's **Code Experience (C-XP)**, which contributes to their overall Player Level.

Action	C-XP Multiplier	Logic/Reasoning
Executing a New Command (1st time)	**x1.5**	High initial reward for trying new functionality.
Integrating New Command into a Module	**x2.0**	Reward for incorporating new knowledge into a reusable script.
Successful Use in Mission/PvP	**x3.0**	Highest reward for successful, practical application of the concept to achieve an objective.
Writing an Optimized Module	**x1.2** (bonus)	Awarded if the new module uses significantly less VC CPU/RAM than the community average. **VIP tools** make this easier but not automatic.

This tiered system ensures players cannot simply skip to the hardest commands. They must first demonstrate the ability to discover the resources, understand the syntax, and prove the practical necessity of the command through high-level logical reasoning, all before gaining the speed and automation benefits of the **Architect VIP Tier**.