That's smart thinking. For a mobile-friendly game, keeping the syntax clean, using minimal special characters, and relying on spaces for separation is key for quick, comfortable input. Here is the revised design for **NexusScript**, tailored for simplicity and mobile typing, incorporating your syntax requirements.

NexusScript Language Design (Mobile-Optimized)

The core philosophy remains: **Everything is an object, and every action is a function/method call.** We'll use spaces to separate command components and dollars signs for variables.

1. Core Syntax and Variable Handling

The syntax is designed to mimic natural language commands while preserving the function of a command-line interface.

Concept	NexusScript Syntax	Example	Function / Mobile Benefit
Command Structure	[Function] [Target] [Parameters]	scan target_ip subnet 24	Minimal special characters, separated by space.
Variable Declaration	set $[variable] = [value]	set $router = new Router(21)	Uses $ to easily denote a variable without complex symbols like % or #.
String / Input	Enclosed in quotes (standard)	set $user = "admin"	Standard string definition.
Object Method Call	$[Object].Method($[params])	$router.connect($user $pass)	Uses familiar dot notation, but uses a space, not a comma, to separate function arguments.
Help System	help [topic]	help scan	Simple, two-word command.

2. Initial Core Commands and Objects

This is the starting set available after the initial boot-up, focusing on local file system and basic network discovery.

Command/Object	Type	NexusScript Syntax Example	Learning Focus
File Listing	Command	ls /v system	Navigating the virtual file system.
File Reading	Command	cat /v logs auth.log	Reading data logs.
Pinging	Command	ping 10 0 0 1	Basic network connection check.
Port Scanning	Command	scan 10 0 0 1	Identifying open services.

Command/Object	Type	NexusScript Syntax Example	Learning Focus
Variable Setting	Command	set $target = new IP 10 0 0 1	Creating an addressable object.
Script Execution	Command	run my first script	Running a player-created module.
Basic Output	Command	print $target	Displaying variable or object data.

3. Simplified Scripting Structure (NexusScript Modules)

Players write modules (functions) to automate sequences. The structure is simple, using bracket characters {} which are easily accessible on mobile keyboards.

Concept	NexusScript Syntax Example	Function
Function Definition	func brute force v1 (target ip)	Defines a reusable module. Arguments are separated by space.
Control Flow (If)	if target.port is open (21) { run ftp login $passfile }	Conditional execution. The condition is wrapped in parentheses ().
Looping	for $line in $wordlist { print $line }	Iterates over items. Requires minimal special characters.
System Calls	system sleep 5	Executes system-level commands (e.g., to pause execution).

Example: A Simple Brute-Force Module

```
func simple brute (target ip) {
    set $wordlist = cat /v data passwords.txt

    # Iterate through the lines of the wordlist file
    for $pass in $wordlist {
        print "Attempting pass: $pass"
        set $login_status = $target.login("admin" $pass)

        # Check if the login was successful
        if $login_status is TRUE {
            print "SUCCESS password found: $pass"
            system break # Exit loop early
        }
    }
}

# Run the script:
run simple brute (10 0 0 5)
```

4. Advanced Tool and Object Design (Scaling)

As the player levels up, new tool-objects are unlocked that leverage the simple syntax but perform complex actions. The complexity is encapsulated *inside* the function, not the syntax.

Unlocked Tool-Object	Method Call (Syntax)	Real-World Concept Simulated
Web Fuzzer	$fuzzer.inject($target "SQL query" $payload)	**SQL Injection/Fuzzing**: The player must provide the target, the method (type of attack), and the data ($payload) to test.
Exploit Framework	$cve.deploy($target "win server v2" $payload)	**Metasploit**: The player uses the Exploit_Object ($cve) against a specific service version on the target.
Packet Sniffer	set $data = $nic.sniff(port 21 duration 30)	**Wireshark/Packet Capture**: Sniffs data for a set duration, then returns a **Data Object** ($data) that requires a subsequent hashcrack command.
Defense Firewall	$firewall.rule add (ip $target action DROP)	**Rule Configuration**: Used in defense missions or to secure the player's own VC, teaching firewall syntax.

5. The Scaling help Command

The help command automatically includes new usage examples and explains the underlying concept as commands are unlocked, functioning as the built-in curriculum.

Advanced State Example (Post-Exploitation)

```
> help $cve.deploy
$cve.deploy
Function: Deploys a loaded Exploit_Object against a specific target
service. Requires a vulnerability ID (CVE Sim).
Usage: $[Exploit_Object].deploy([target IP] [target OS] [payload
module])
Example: $exploit_v3.deploy($server "Linux Kernel 4.19"
$shell_payload)
Concept: Exploit Payloads and Targeting (similar to Metasploit).
```

This design keeps the core scripting logic simple and easy to type, making it accessible on a mobile device, while reserving complexity for the logical structure of the player's modules and the strategic use of advanced, mission-unlocked objects.