with-qwen

give me principles of "suckless neovim propaganda"

elaborate further in two parts, start with the first

then the rest of them

✨ Suckless Neovim Propaganda ✨

🚀 Why Suffer? Embrace the Suckless Philosophy for Your Editor.

Tired of feature bloat, endless configuration files, and editors that feel like a sluggish operating system? Join the movement that values **simplicity, clarity, and speed** above all else.

💡 What is Suckless?

It's a principle, not just a set of tools. It means:

- **Minimalism:** No features you don't need. The source code fits in your head.
- **Clarity:** Code that is easy to audit, understand, and modify.
- **Efficiency:** Tools that are lightning-fast because they are lightweight.

⚡ Neovim: The Suckless Editor (If You Configure It Right)

While Neovim is technically feature-rich, its power comes from its **extensibility** and **sane defaults**.

- **Forget VSCode/Emacs:** Stop installing 500 plugins just to get basic functionality.
- **The *Actual* Goal:** Use Neovim's built-in power (`lsp` , `treesitter` , etc.) combined with **only essential Lua plugins** to build a tailored, minimal, and *fast* environment.
- **Your Editor, Your Source Code:** Control every byte of your configuration. If you don't know what a plugin does, **delete it.**

Feature	Bloated Editor (The Enemy)	Suckless Neovim
Startup Time	Slow (Loading electron/megabytes of plugins)	**Instantaneous**
Configuration	Complex, multi-layered, opaque	Single, clean `init.lua`
Memory Use	Gigabytes	**Megabytes**
Philosophy	Add, add, add	**Subtract, simplify, conquer**

🛠️ How to Go Suckless in Neovim

1. **Delete Your Plugin Manager:** (Optional, but very Suckless) Use built-in packages or a simple, fast manager like `lazy.nvim` —but use it *sparingly*.
2. **Learn the Built-ins:** Master the internal LSP, `telescope.nvim` (only one major fuzzy finder!), and Neovim's keybindings.
3. **Lua Everything:** Write all configuration in Lua. It's fast, simple, and the native way.
4. **One Tool, One Job:** Don't install five file explorers. Pick one great one (`nvim-tree.lua` or built-in netrw) and stick to it.
5. **Embrace the Terminal:** Use external terminal tools for tasks (like grep/find) instead of heavy GUI-style plugins.

> *"A tool should be as simple as possible, but no simpler." - Slightly adapted Suckless Proverb*

🏁 Conclusion: Be the Master of Your Machine.

Stop being a victim of someone else's idea of a perfect editor. Stop the bloat. **Simplify. Master Neovim.**

#suckless #neovim #minimalism #speed

That's a great request! I'll elaborate further on the **"Suckless Neovim Propaganda"** in two distinct parts.

Here is **Part One: Deep Dive into the Suckless Philosophy and Neovim's Alignment.**

🧘 Part One: The Philosophy of Suckless and Neovim's Minimalist Alignment

The "Suckless" philosophy is often associated with the developers of the *suckless.org* software collection (like **dwm** and **st**), but it represents a broader principle that directly opposes the modern trend of software bloat.

📜 The Core Tenets of Suckless

The philosophy is governed by several strict, counter-cultural principles:

1. **Simplicity Over Complexity:** Suckless software strives to be **minimal**. It aims to solve *one problem* well and avoids creeping feature-ism (or *featuritis*). This means minimal codebases that are easy for the user to read, understand, and modify. If a feature can be accomplished better by another existing tool (like using an external file manager instead of embedding one), it should be omitted.
2. **Clarity and Transparency:** The source code should be small enough to "fit in your head." This low line count (LoC) reduces bugs, improves security (as the code is easy to audit), and makes the software **truly free**—free to be modified by the user.
3. **Use External Tools:** Rather than being a monolithic application that tries to do everything, Suckless tools often rely on the **Unix philosophy**—connecting small, single-purpose utilities together via pipes. This is a core reason why tools like **dwm** rely on external shell scripts for status bar information.

🧠 How Neovim Embodies This (or Can Be Forced To)

Neovim, due to its history and architecture, is an excellent candidate for a Suckless-compliant editor, provided the user adopts a disciplined approach.

- **The Power of Built-ins:** The most "Suckless" Neovim configuration is the one that uses the **least number of external plugins**. Neovim now ships with powerful, native components:
    - **Built-in LSP (Language Server Protocol):** Eliminates the need for monolithic language-specific plugins. The user only needs to configure the server and client setup once.
    - **Tree-sitter Integration:** Provides superior syntax highlighting and structural navigation, often replacing numerous older, slower regex-based highlighting schemes.
    - **Native Lua Configuration:** Lua is fast, lightweight, and directly integrated, making configurations much quicker to load than older Vimscript files.
- **The Configuration as Source Code:** A Suckless Neovim user treats their `init.lua` (or `init.vim`) not as a temporary script, but as a piece of **essential source code**. Every line must be justified. A user should be able to look at their config and know exactly what every line and every *required* plugin does.

- **Minimizing the Plugin Interface:** Instead of installing plugins for every small task, the Suckless approach dictates leveraging Neovim's core features (like registers, macros, and command-line execution) and using the terminal for external tasks:
    - Need to find files? Use `find` or `grep` in a terminal pane, not a slow, embedded file-search tool.
    - Need to view a file structure? Use the built-in `netrw` or a simple, fast tree-like utility, avoiding heavy GUI-styled explorers.

By following this minimal configuration path, you transform Neovim from a potentially heavy editor into a **highly optimized, bespoke tool**—a true manifestation of the Suckless ethos in an editing environment.

Would you like me to continue with Part Two, which will focus on specific configuration techniques and plugin choices to achieve this Suckless state?

Absolutely! Here is **Part Two: Specific Configuration Techniques and Plugin Choices for a Suckless Neovim Setup.**

🛠️ Part Two: Techniques for a Truly Suckless Neovim Configuration

Achieving a "Suckless" editor experience requires discipline in how you choose and manage your Neovim extensions. It's about being **intentionally lean** and prioritizing the terminal's built-in power.

🚫 The "Don't Install" List

The first step to Suckless is subtraction. Avoid plugins that duplicate functionality better served by the terminal or Neovim's built-ins:

- **GUI-Style File Explorers:** Avoid heavy, mouse-centric tree plugins. Learn and use `:Ex` (the built-in `netrw`), or if you must, use a simple one like `nvim-tree.lua` only for projects.
- **Built-in Task Runners:** Use `make` or shell scripts run directly from a Neovim terminal (`:term`) or through simple mappings (e.g., `:!make`).
- **Non-Essential Eye Candy:** Avoid massive status-line generators, excessive animations, or tab-line managers. Keep it simple with the built-in status line or a minimal utility like `lualine.nvim`.

⚙️ Plugin Discipline (If You Must)

If a task cannot be reasonably done with core Neovim, only choose plugins that follow the Suckless spirit: **small, fast, and Lua-based.**

Category	Suckless Choice	Why It Works
Fuzzy Finder	`telescope.nvim`	One powerful tool for all searching (files, greps, commands). Fast and native.
Plugin Manager	`lazy.nvim`	Extremely fast and allows for conditional loading, keeping startup instantaneous.
Completion/Snippet	`nvim-cmp`	Modular and essential for LSP use, but only install *necessary* sources.

Git Integration	`fugitive.vim`	A classic, minimal interface for Git that relies on Vim commands, not heavy pop-ups.
Syntax Highlighting	`nvim-treesitter`	The native, fast solution that replaces bulky regex-based syntax files.

🚀 Configuration Techniques

A Suckless Neovim configuration is **imperative**—it explicitly defines exactly what the user wants.

1. **Use Conditional Loading:** If you use a plugin manager like `lazy.nvim`, use its capabilities to only load plugins when they are actually needed (e.g., only load a specific language server when opening a file of that type). This ensures **zero overhead** on general startup.
2. **Mapping Over Commands:** Set up efficient keymaps that chain built-in commands together. For instance, mapping a single key to compile and then execute a script:
   > *`:nnoremap <leader>r :w <bar> !./run_script.sh<CR>`*
3. **Lean and Clean `init.lua`:** Structure your configuration into small, well-named files (e.g., `mappings.lua`, `lsp.lua`, `plugins.lua`). Crucially, your top-level `init.lua` should be a **manifest** that simply loads these modules, making the entire setup transparent and easy to audit.

💡 The End Result: Pure Velocity

By adhering to these rules, your Neovim configuration becomes a true example of high performance through subtraction. You gain:

- **Near-Instant Startup:** No more waiting for hundreds of plugins to initialize.
- **Total Control:** Every feature in your editor is there because *you* decided it should be.
- **Minimal Resource Use:** Your editor runs efficiently, allowing your system resources to be dedicated to compilation or computation.

The Suckless way is about maximizing **user productivity** by minimizing software complexity. It's the ultimate customization through **deletion**.

Would you like me to find a specific video showing a very minimal, Suckless-style Neovim configuration?
