Audion System Manager — user guide

Русский

Contents

- Starting
- The window
- POWERSHELL 7
- TERMINAL
- PYTHON
- VS CODE
- GIT
- AI CLI
- DOWNLOADS
- The folder of a portable build
- Where things are
- Keys
- Network and privacy

Audion System Manager is one window for the developer environment on Windows: **PowerShell 7**, **Windows Terminal**, **Python 3.12, 3.13 and 3.14**, **VS Code**, **Git** with **GitHub CLI**, and the agents of the console — **Claude Code**, **Codex**, **Grok Build**. Each program can be installed for the system or unpacked as a portable folder — with every option a package manager does not give, and with the Audion settings. Everything is done with buttons: no black console window, no command-line switches.

Starting

Start **Start.exe**. The program runs with the rights of an ordinary user. Administrator rights are needed only for an install for the whole machine (PowerShell 7, Python for all users, VS Code "for the machine"): Windows asks for them with a UAC prompt at the moment the installer starts, and only for it. Declined — nothing is installed and nothing changes. Downloads, portable builds, the settings of the terminal and the editor, the extensions need no rights.

The language is taken from the system at the first start (Russian on a Russian Windows, English on any other). **RU | EN** at the top right switch it, the moon and the sun switch the theme. The font size (**a | A | A+**) is in **ABOUT**, the **SETTINGS** card. The choice is remembered.

A second start from the same folder does not open a second copy but shows the window of the running one: two installers at once would get in each other's way.

The window

The top row — two dark pieces: the six sections, and the language and theme on the right. The name and version of the program are in the title bar and in ABOUT.

The sections — POWERSHELL 7, TERMINAL, PYTHON, VS CODE, GIT, AI CLI. Each has a colour of its own; the colour of the line under the header shows which one is chosen. Under the name of a section — the version in the mode chosen on its page and the mark of the mode (a screen — for the system, a stick — portable). The dot by the version: green — the latest version, orange — not installed or a newer one is out ("1.132.0 → 1.142.0"), gray — the source did not answer. AI CLI shows a summary instead of a version, "2 of 3 · signed in 2": how many agents are installed and how many are signed in.

The page of a section begins with the mode switch — two buttons in one frame at the top left:

- **FOR THE SYSTEM** — the program is installed on the machine (or into the profile, where that is the idea) with its options;
- **PORTABLE** — the program is unpacked into a folder and installs nothing; the folder can be carried to another machine.

Under it — the cards of the chosen mode in two columns with a line between them; both columns belong to the mode chosen by the switch. On the left, what there is: the state (what is installed, where, which version is the latest, what lies in the downloads). On the right, what to do with it: the install options, and in the portable mode the folder and **UNPACKING** (how the build is unpacked from the archive in Download, its buttons). The cards both modes share — the PowerShell profile, the terminal font, the VS Code settings and extensions, GitHub CLI and the Git settings — stand in the same

columns. In AI CLI the agents go left and right by turns; in ABOUT the program is on the left, the settings, keys and licence on the right. In a narrow window the columns stand one under the other: the left one first, then the right one.

At the bottom, always:

- the status line: **F1 DOCS** (this guide), the last message, the progress of an operation with **STOP**, and **JOURNAL**;
- the tools **DOWNLOADS** and **ABOUT** (press again to go back to the section); ABOUT holds, besides the facts of the program, the **SETTINGS** card: the font size, the version check at start, the folder of the Audion settings;
- on the right the two main buttons for the section chosen on top, in the mode chosen on its page: **DOWNLOAD** (the latest version into the Download folder; gray — the latest is downloaded already) and **INSTALL** / **UNPACK** / **UPDATE** (install or unpack with the options of the page; nothing downloaded — downloads first). The line under a button says what exactly it will do.

The journal — at the bottom above the buttons; the bar between the page and the journal is dragged with the mouse, and the height of the journal is remembered. Everything the program does is in it, with the output of the installers, pip, WinGet and VS Code. It opens by itself when an operation begins; **Ctrl+J** or **JOURNAL** hide and show it. The journal file of the day and the logs of the installers (msiexec, python.org, VS Code) are in `._runtime\logs`.

One long operation at a time: while a download or an install runs, the buttons that would start another are gray. **STOP** stops a download, an unpacking, the install of extensions. An installer running with administrator rights cannot be stopped: an install broken off half-way is worse than one let finish.

POWERSHELL 7

For the system. PowerShell 7 is installed from the MSI of the GitHub release — the same package WinGet installs, but with the options WinGet does not give. The options are the chips of the **INSTALL OPTIONS** card (amber — on):

Option	What it gives
OPEN HERE	"Open PowerShell 7 here" in the menu of folders
RUN .PS1	"Run with PowerShell 7" in the menu of .ps1 files
PWSH IN PATH	the `pwsh` command in any console

Option	What it gives
EVENT LOG	the messages of PowerShell in Event Viewer
MICROSOFT UPDATE	updates through Microsoft Update and WSUS — PowerShell is updated together with Windows
REMOTING	`Enable-PSRemoting`: opens a WinRM listener, so it stays off until you switch it on
NO TELEMETRY	telemetry off; if it was turned off earlier, the program sees that and does not turn it back on

The choice survives updates: the package keeps it and puts it back at the next install, whoever does it — Microsoft Update or WinGet. What is on now the program reads where the installer itself writes (`HKLM\SOFTWARE\Microsoft\PowerShellCore\InstallerProperties`); the line under the chips says what **INSTALL** will change.

INSTALL takes the newest MSI from Download (none — downloads it), checks its SHA-256 against the release and starts msiexec — Windows asks for administrator rights. The same version is installed already — adds the options to it; everything is so already — does nothing. **REINSTALL** installs the package again over it.

Portable. The zip of the same release is unpacked into a folder (`Portable\PowerShell` beside the program by default; the **Folder** field changes it). The portable PowerShell takes the user's profile and modules from Documents, as the installed one does. **START** opens it in a console of its own.

The PowerShell profile is shared by both modes: the script `Documents\PowerShell\Microsoft.PowerShell_profile.ps1` that runs at the start of every window. The Audion profile (`Presets\PowerShell`) — UTF-8 encoding, PSReadLine suggestions from the history as a list, history search with the arrows. **CARRY OVER PROFILE** puts it there (the previous one is set aside with the date in its name), **TAKE FROM THE MACHINE** makes the profile of this machine the preset. With the **Author's settings: the Audion profile** box (ticked by default) the profile is put at every install and unpacking too; without it PowerShell 7 goes in clean and the profile on the machine is left alone.

TERMINAL

For the system. Windows Terminal from the Microsoft Store. Windows 11 has it already and the Store updates it; when it is missing or old, the button by the version installs or updates it through WinGet (the same package, for the user, no administrator rights).

The **AUDION SETTINGS** card shows what is in the settings file (`Presets\Terminal\settings.json`: the Audion scheme with the orange cursor #FF9807, the Russian interface, three keyboard shortcuts, PowerShell by default, the Cascadia Mono font) and where it goes: the terminal keeps its settings in different places depending on where it came from (Store, Preview, unpacked). **CARRY OVER SETTINGS** puts the file into every one found; the previous settings are set aside beside them with the date in the name — an ordinary copy brings them back. The file is checked for valid JSON first. If the terminal is open, better close its windows: on exit it may write its state over them. **TAKE FROM THE MACHINE** makes the settings of this machine's terminal the preset — so the preset keeps up with what was changed in the terminal itself.

Default terminal (Windows 11) — the window console programs open in: **WINDOWS TERMINAL**, **WINDOWS CONSOLE** or **LET WINDOWS DECIDE**. Kept in the user registry, no rights needed. A portable Windows Terminal cannot be the default.

Font — Cascadia Code and Cascadia Mono with the NF (Nerd Font: icons in prompts and file lists) and PL variants from GitHub, for the user, no rights. A face Windows has already (Cascadia Mono in Windows 11) is not touched; the same files are not installed twice.

Portable. The zip of the GitHub release is unpacked into a folder (`Portable\Terminal` by default), a `.portable` file is put beside WindowsTerminal.exe — and the terminal keeps its settings in `settings\` beside itself. With the **Author's settings** box the Audion settings go there too; the same box in the **AUDION SETTINGS** card decides whether they go into the installed terminal after an install. Ticked by default; without it the terminal goes in clean, with its own settings. On an update `settings\` and `.portable` stay.

PYTHON

Under the title of the page — **Version**: 3.12, 3.13 or 3.14. The versions stand side by side, each in a folder of its own, with its own installer, archive and latest version at python.org; `py -3.12`, `py -3.14` pick one, `py` without a number the newest one installed. 3.12 is the default: the embedded runtimes of the Audion projects are on it (the wheel tag cp312). The choice is one for both modes and is kept in the settings of the program.

For the system. The python.org installer with keys of its own — for all users, into a folder of its own (`C:\Python312`, `C:\Python313` or `C:\Python314` by default — every version has its own: in Program Files every `pip install` runs into an elevation prompt, in its own folder it does not). The keys are the chips of the **INSTALL** card: the path at the front of the system PATH (the Store stub no longer catches `python`), the `py` launcher, tcl/tk; .py associations, Start menu shortcuts, the documentation and the tests are off by default. After the installer the program removes the Store stubs

(zero-size python.exe and python3.exe in the profile) and puts a copy of python.exe beside it as python3.exe, checks which python and python3 answer in a new console and updates pip. The stubs are the aliases of App Installer, and Windows brings them back with every update of it; but python3 is then answered by the real Python — its folder stands earlier in PATH than the folder of the stubs. The **Store stubs** line says whether they are shadowed.

Why 3.12: the runtimes inside the Audion projects are 3.12, and the wheel tag (cp312) has to match. After 3.12.10 python.org releases 3.12 as sources only, so the latest version here is the latest one with a Windows installer.

The **IN THE SYSTEM** card shows what usually breaks: every Python the registry knows (even those not in PATH), the py launcher, which python answers in a console, and the Store stubs. **UPDATE PIP OF ALL** updates pip of every Python the py launcher knows (where writing needs rights, Windows asks for them).

Portable. python.org's full archive (`python-3.N.M-amd64.zip`, the one the Python Install Manager takes) is unpacked into the folder of its version (`Portable\Python` for 3.12, `Portable\Python313` and `Portable\Python314` for the others by default): the whole Python with pip, nothing installed, nothing written to the registry. The Doc folder is not unpacked unless the **DOCUMENTATION** chip is on. On an update your packages (`Lib\site-packages` and `Scripts`) move to the new build. The py launcher does not see a portable Python — start it by the full path of python.exe or with **START**.

VS CODE

For the system. Microsoft's installer — **FOR THE MACHINE** (for all users, needs administrator rights) or **INTO THE PROFILE** (no rights, only for you) — with every box of its wizard: the menus of files and folders, associations, `code` in PATH, a desktop shortcut. Then the Audion settings and the extensions. The same or a newer version is installed already — the installer is not run, the settings and the extensions are done. The installer replaces the program, so VS Code has to be closed.

Portable. The archive is unpacked into a folder (`Portable\VSCode` by default), with a `data` folder beside Code.exe where the editor keeps its settings and extensions and writes nothing to the profile. The portable VS Code does not update itself — this program updates it: a fresh archive, the program replaced, `data` untouched.

Settings and extensions are shared by both modes. The settings are the file `Presets\VSCode\settings.json`; trust in folders is off in it (`security.workspace.trust.enabled`), so the editor does not ask about every folder. Whether they go in, and **ripgrep for Todo Tree** with them, is chosen in the window of the choice (below): Todo

Tree finds its marks with ripgrep, whose path inside VS Code changes with every update, so a separately installed one is taken (none — WinGet installs it) and its path is written into the settings. The same **Author's settings** box stands right on the page too, in the card of the install of each mode.

TAKE FROM THE MACHINE makes the settings, keys (`keybindings.json`) and snippets of this machine's VS Code the preset — the installed or the portable one, by the mode of the page; the machine's path of ripgrep is left out. The keys and snippets of the preset go along with the settings. **UPDATE EXTENSIONS** updates every extension now by VS Code itself (`code --update-extensions`); usually they update by themselves too — `extensions.autoUpdate` is on in the Audion settings.

The extensions come from three lists in `Presets\VSCode`: **CORE** (`extensions.txt`, installed always), **AI AGENTS** (`extensions-ai.txt`: Claude, Codex and Kilo Code by default; Grok by request, a panel over a separate Grok Build CLI) and **BY REQUEST** (`extensions-optional.txt`). Kilo Code takes the models of many providers by an API key or through the Kilo gateway, and the Codex and Grok subscriptions connect to it officially; do not connect a Claude subscription to it — Anthropic does not allow working by it through third-party agents and bans the account for it (Claude in Kilo Code — by an API key only).

What goes in is ticked in a window of its own: on top **Author's settings** and **ripgrep for Todo Tree**, below every extension of the three lists by sections, the buttons **ALL**, **BY THE LISTS** (as the lists mean it) and **CLEAR ALL**, and a count of what is ticked and installed already. The window opens before **INSTALL**, **REINSTALL** and **UNPACK** (its button below starts the install), before **EXTENSIONS ONLY** (without the settings) and by **CHOOSE EXTENSIONS** (the **SAVE** button) — it stands in the card of the install beside the count of what is ticked, under the **Author's settings** box. The ticks are kept in the settings of the program when the window is left by its button; **CANCEL** leaves everything as it was. The tip of a section's name says why exactly these are in it, the tip of a tick gives the full identifier and whether the extension is installed. The lists can be edited (the **LISTS** button): one line — one identifier, a heading `# --- Name ---` opens a section.

GIT

For the system. Git for Windows from the GitHub release for the whole machine, with the boxes of its installer's wizard: Git in PATH, the VS Code editor, branch main, CRLF line endings, Git Credential Manager, Windows certificates, the file system cache, symbolic links (off), the Explorer menu, Git LFS, associations, a Git Bash profile in Windows Terminal, a desktop shortcut (off). Each box goes to the installer as its own key. The latest is installed already — the installer is not run; **REINSTALL** changes the boxes of an installed Git. Needs administrator rights.

Portable. PortableGit of the same release unpacks itself beside the folder (`Portable\Git` by default) and takes its place: Git, Git Bash and all it needs in one folder.

GitHub CLI (`gh`) is installed in the mode of the page: the MSI of the release for the system or the zip into `Portable\GitHub CLI`, checked against the checksums of the release. **SIGN IN** opens `gh auth login --web` in a console window of its own: gh shows a one-time code, entered in the browser on github.com — after that both `gh` and Git go to GitHub with this sign-in. **SIGN OUT** — `gh auth logout`.

Git settings — the preset `Presets\Git\gitconfig` (line endings, long paths, Cyrillic in file names, the editor, branch main, UTF-8). **CARRY OVER SETTINGS** puts the keys into the global settings one by one (`git config --global`): the name, e-mail and sign-in in `~\.gitconfig` stay. The **Author's settings: the Audion gitconfig** box (ticked by default) does it after every install and unpacking too; without it Git goes in clean. Beside each key is what is in force for you now. **TAKE FROM THE MACHINE** makes the global settings of this machine the preset — without the personal and machine ones (user, credential, url, http and anything that looks like an address with a login or a token).

AI CLI

The agents of the console: **Claude Code**, **Codex**, **Grok Build** and, last, **Antigravity CLI** (`agy`, Google's agent that succeeded Gemini CLI). The section has no modes: the agents live in the user profile and install with no administrator rights.

Each agent shows its version against the latest, its sign-in and its buttons:

- **INSTALL / UPDATE** — with its authors' installer (PowerShell, `irm … | iex`) as their documentation gives it; an installed one is updated with its own command (`claude update`, `grok update`, `agy update`; Codex — with the same installer). The folder of the program is added to the user PATH when the installer did not do it.
- **SIGN IN** — OAuth in the browser with your subscription: the CLI's own sign-in command runs in a console window of its own (`claude auth login`, `codex login`, `grok login --oauth`). Antigravity CLI has no sign-in command of its own: `agy` itself starts, and with no saved sign-in it opens Google's sign-in page.
- **SIGN OUT** — sign out of the account (after a question). Antigravity CLI has no such button: sign out with the `/logout` command in `agy` itself.
- **START** — the agent in a console window of its own.

Whether an agent is signed in the program sees only by the file of its sign-in being there; the file itself is never read. Antigravity CLI keeps its sign-in in the Windows Credential Manager — the program

does not check it. **INSTALL** at the bottom installs and updates every agent ticked with **WITH THE BUTTON BELOW**; Antigravity CLI is not ticked by default and is not counted in the summary of the section on top until it is installed.

DOWNLOADS

What is downloaded into the `Download` folder beside the program: installers and portable archives, their size and date. The sections take the newest file of their kind from it and remove the previous one of the same kind when a new one is downloaded. A file from the folder is checked before an install: against the published SHA-256 where the source gives one, else against the sum written down at its download; one that does not match is removed and downloaded again. The folder can be carried to another machine with the program — installing then needs no network; a file changed since its download is not taken without the network. **TO THE RECYCLE BIN** and **ALL TO THE RECYCLE BIN** move files to the Recycle Bin.

The folder of a portable build

The **Folder** field of the portable mode takes a path on Enter or when you leave the field; **CHOOSE...** opens a folder picker, **DEFAULT** goes back to `Portable\<program>`. An empty folder will do, or a folder where this program is unpacked already: an update replaces its files, so a folder of other things is never touched. The root of a drive will not do. The new build is unpacked beside and takes the place of the folder whole; when something fails (a file of the folder is open, say), the previous build stays with all its data. **TO THE RECYCLE BIN** moves a portable build to the Recycle Bin (with its settings), not while it runs.

Where things are

Path	What is there
`Start.exe`	starts the program
`App\`	the program itself (its own .NET runtime, nothing to install)
`config\settings.json`	your choices: language, theme, modes, folders, options, the marks of extensions
`Presets\`	the settings of Windows Terminal, VS Code (with keys and snippets) and Git, the PowerShell profile, the lists of extensions — what the program carries over

Path	What is there
`Download\`	downloaded installers and archives
`Portable\`	portable builds (unless another folder is named)
`._runtime\logs\`	the journal by days and the logs of the installers

Keys

Key	What it does
F1	this guide
Ctrl+1 … Ctrl+6	the sections POWERSHELL 7, TERMINAL, PYTHON, VS CODE, GIT, AI CLI
Ctrl+7, Ctrl+8	DOWNLOADS, ABOUT
Ctrl+J	the journal
Esc	back to the section

Network and privacy

The program talks only to the sources of the programs themselves: GitHub (PowerShell, Windows Terminal, Git, GitHub CLI, Cascadia), python.org, Microsoft's update service (VS Code), the Visual Studio Marketplace (extensions — through VS Code itself), WinGet and the installers of the agents (claude.ai, chatgpt.com, x.ai). The answer about versions is kept for six hours; the **CHECK VERSIONS AT START** chip in ABOUT switches the check at start off. Nothing is collected and nothing is sent anywhere.