

EasyEngine CLI

EasyEngine **CLI** is a command-line tool for managing **WordPress sites** on a server—mainly on **Linux**—by automating common tasks like setup, configuration, and maintenance. It’s often used to quickly create and manage WordPress stacks (e.g., with **Nginx**, **PHP**, **MariaDB/MySQL**, and sometimes **Docker**) using simple terminal commands. ☐

- [EasyEngine CLI — Course Outline for a WordPress Developer](#) ☐☐
- [EasyEngine CLI Cheat Sheet](#) ☐☐
- [Terminal “GUI” for EasyEngine \(Debian\)](#) ☐☐☐☐
- [c1\) Orientation: What EasyEngine is \(and why it changes your workflow\)](#) ☐☐
 - [c1.1 — What EasyEngine CLI is \(and why it’s worth switching\)](#) ☐☐

EasyEngine CLI — Course Outline for a WordPress Developer ?

Below is a **chaptered, addressable syllabus** you can reference like `c2.6` (Chapter 2, Subsection 6). It's designed around what you shared from the GitHub docs (site management for **HTML/WP**, **cron**, and **shell**) and also adds the *practical, WordPress-on-a-root-server* knowledge you'll want alongside those commands. ☐

c1) Orientation: What EasyEngine *is* (and why it changes your workflow) ?

c1.1 — EasyEngine mental model (shared hosting vs root server vs containers)

c1.2 — What EasyEngine manages for you (sites, containers, services, SSL)

c1.3 — Core CLI structure: `ee <noun> <verb> [args] [options]`

c1.4 — Local machine vs server: where you run `ee` and what it touches

c1.5 — Learning map: what you'll be able to do by the end of the course

c2) Site Lifecycle Fundamentals (HTML + WordPress) ??

c2.1 — Naming and identifying sites (`<site-name>` conventions, DNS expectations)

c2.2 — Creating sites: the shared mechanics across types

- What `--type` means conceptually
- When `--skip-status-check` matters and when it's risky

c2.3 — `ee site create --type=html`

- c2.3.1 Minimal create
- c2.3.2 Create with SSL (`--ssl=le`, `--ssl=self`)
- c2.3.3 Wildcard SSL (`--wildcard`) and when you actually need it

c2.4 — `ee site create --type=wp` (WordPress install paths)

- c2.4.1 Minimal WP create
- c2.4.2 WP + Let's Encrypt SSL
- c2.4.3 WP + wildcard SSL
- c2.4.4 WP + Redis cache (`--cache`) and what it implies
- c2.4.5 WP multisite (`--mu=subdir` vs `--mu=subdom`)
- c2.4.6 VIP flow (`--vip`) and using a repo containing `wp-content`
- c2.4.7 Skipping steps (`--skip-install`, `--skip-content`) and why/when
- c2.4.8 Version selection (`--version=latest|nightly|x.y.z`)

c2.5 — Managing a site's runtime state

- c2.5.1 `ee site enable` (and what "enable" means in practice)
- c2.5.2 `ee site disable` (what stops/removes; what persists)

- c2.5.3 `--verify` and dependent “global services”
- c2.5.4 `--force` usage patterns (and cautions)

c2.6 — Updating an existing site: `ee site update`

- c2.6.1 Adding SSL to a non-SSL site (`--ssl=le|self`)
- c2.6.2 Switching to wildcard (`--wildcard`)
- c2.6.3 Safe operational routine (pre-checks, downtime avoidance mindset)

c2.7 — SSL operations: `ee site ssl`

- c2.7.1 Challenge verification concepts
- c2.7.2 Renewal behavior and `--force`
- c2.7.3 Practical renewal troubleshooting checklist

c2.8 — Inspecting & inventorying: `ee site info` and `ee site list`

- c2.8.1 `ee site info` output: credentials, links, what to look for
- c2.8.2 `ee site list` filters (`--enabled`, `--disabled`)
- c2.8.3 Output formats (`--format=table|csv|yaml|json|count|text`)

c2.9 — Reload/restart: `ee site reload` and `ee site restart`

- c2.9.1 Reload vs restart (what you use each for)
- c2.9.2 “Type” variants in docs (`--type=html|wp`) and expected behavior

c2.10 — Deleting sites: `ee site delete`

- c2.10.1 Confirmation and `--yes`
- c2.10.2 Data safety: what to back up *before* deletion

c2.11 — Caching operations: `ee site clean`

- c2.11.1 Clearing both caches (default behavior)
- c2.11.2 `--page` vs `--object` (when to clear which)

c2.12 — Sharing a site online: `ee site share` (ngrok)

- c2.12.1 Create a share link
 - c2.12.2 `--refresh` behavior
 - c2.12.3 Taking it down (`--disable`)
 - c2.12.4 Using `--token` (ngrok token hygiene)
-

c3) WordPress Provisioning Details (Deep Dive) ?

(Still based on your doc excerpt, but expanded into the decisions you'll actually make.)

c3.1 — Admin bootstrap options (`--title`, `--admin-user`, `--admin-pass`, `--admin-email`)

c3.2 — PHP version selection (`--php=5.6|7.2|latest`) and compatibility strategy

c3.3 — Database configuration patterns

- c3.3.1 Defaults vs explicit config (`--dbname`, `--dbuser`, `--dbpass`, etc.)
- c3.3.2 Remote DB host (`--dbhost`) implications
- c3.3.3 `--skip-check` and when it can bite you
- c3.3.4 `--force` and “reset remote database if not empty”

c3.4 — Local DB & local Redis containers (`--local-db`, `--with-local-redis`)

c3.5 — Multisite operations mindset (subdir vs subdom prerequisites)

c3.6 — VIP workflow (`--vip`) as a deployment pattern

c4) Cron Management with `ee cron` ??

c4.1 — What EasyEngine cron controls (site-level vs host-level)

c4.2 — Listing jobs: `ee cron list [<site-name>] [--all]`

c4.3 — Creating jobs: `ee cron create`

- c4.3.1 Site cron vs host cron (`<site-name>` vs `host`)
- c4.3.2 `--command` quoting rules and practical examples
- c4.3.3 `--schedule` formats:
 - Classic Linux cron strings
 - Macros like `@weekly`, `@daily`, etc.
 - `@every <duration>` (e.g., `@every 10m`, combined units)

c4.4 — Running immediately: `ee cron run-now <cron-id>`

c4.5 — Updating jobs: `ee cron update <id>`

- c4.5.1 Changing site, command, schedule, and user

c4.6 — Deleting jobs: `ee cron delete <cron-id>`

c4.7 — WordPress-specific cron recipes (WP-Cron alternatives, due-now runners)

c5) Shell Access with `ee shell` (Your daily driver) ?

c5.1 — Interactive shell into a site container: `ee shell <site-name>`

c5.2 — Choosing the user: `--user=root|www-data|...` (practical safety rules)

c5.3 — Choosing a service: `--service=php` (default) vs `--service=nginx`, etc.

c5.4 — Non-interactive execution: `--command='...'`

c5.5 — TTY considerations: `--skip-tty` (when automation needs it)

c5.6 — Common WP developer workflows via `ee shell`

- Running WP-CLI operations
- Inspecting logs/configs
- Quick Nginx tests/reloads

c6) Operational Routines (Putting commands into real workflows) ?

c6.1 — “Day 1” routine: create ? SSL ? verify ? info ? share (optional)

c6.2 — “Release” routine: maintenance mode, cache clear, reload/restart

c6.3 — “Incident” routine: disable/enable, verify dependencies, SSL renewal

c6.4 — Safe defaults vs dangerous flags (`--force`, `--skip-check`, `--skip-status-check`)

c6.5 — Auditing & reporting with `ee site list --format=...`

c7) WordPress Performance & Architecture Tuning (EE-centric mindset) ?

c7.1 — Page cache vs object cache: what you clear and why

c7.2 — Redis in practice (`--cache`, `--with-local-redis`)

c7.3 — PHP version strategy for client projects

c7.4 — Multisite scaling considerations

c7.5 — When to prefer local DB container vs global/remote DB

c8) Security & Access Patterns (Root server reality check) ?

c8.1 — SSL choices (`le` vs `self`) and practical environments

c8.2 — Principle of least privilege for `ee shell`
`--user=...`

c8.3 — ngrok sharing: token handling and exposure control

c8.4 — Deletion safeguards and backup checkpoints

c9) Capstone Projects (guided practice paths) ?

c9.1 — Build a production-like WP site with SSL + cron runner + caching

c9.2 — Build a multisite (subdir/subdom) with correct SSL and cron strategy

c9.3 — VIP-style site: `--vip` repo workflow + operational commands

c9.4 — Maintenance toolkit: scripts using `ee shell --command, ee site list --format=json`

c10) Quick Reference Index (command ?topic map) ?

1. Site

1. c2.3 — `ee site create --type=html`
2. c2.4 / c3 — `ee site create --type=wp`
3. c2.10 — `ee site delete`
4. c2.6 — `ee site update`
5. c2.5 — `ee site enable|disable`
6. c2.7 — `ee site ssl`
7. c2.8 — `ee site info|list`
8. c2.9 — `ee site reload|restart`
9. c2.11 — `ee site clean`
10. c2.12 — `ee site share`

2. Cron

1. c4.3 — ee cron create
2. c4.2 — ee cron list
3. c4.5 — ee cron update
4. c4.6 — ee cron delete
5. c4.4 — ee cron run-now

3. Shell

1. c5 — ee shell
-

Next step ?

Tell me where you want to start:

1. `c2.4` (WordPress site creation end-to-end), or
2. `c5` (shell workflows; fastest way to feel productive), or
3. Name any code like `c2.6` and I'll "execute" that chapter in lesson format.

EasyEngine CLI Cheat Sheet ?

A quick-reference guide to the most common **site**, **WordPress**, **cron**, and **shell** workflows.

// Conventions

1. Replace `example.com` with your domain (site name).
2. Square brackets `[...]` mean *optional*.
3. Angle brackets `<...>` mean *required*.
4. SSL values you'll see here: `le` (Let's Encrypt) and `self` (self-signed).

Site Management (All Site Types) ??

1) Create a site (HTML)

Command

```
ee site create --type=html <site-name> [--ssl=<value>] [--wildcard] [--skip-status-check]
```

Options

1. `<site-name>`: Domain / site name.
2. `--ssl=<value>`: Enable SSL.
 1. `le` = Let's Encrypt
 2. `self` = Self-signed
3. `--wildcard`: Request a wildcard certificate (typically with Let's Encrypt).
4. `--skip-status-check`: Skip checking site status before action.

Examples

1. Create an HTML site:

```
ee site create example.com --type=html
```

2. Create with Let's Encrypt SSL:

```
ee site create example.com --type=html --ssl=le
```

3. Create with Let's Encrypt wildcard SSL:

```
ee site create example.com --type=html --ssl=le --wildcard
```

4. Create with self-signed SSL:

```
ee site create example.com --type=html --ssl=self
```

2) Delete a site ?

Command

```
ee site delete <site-name> [--yes]
```

Options

1. `<site-name>`: Site to delete.
2. `--yes`: Skip confirmation prompt.

Example

```
ee site delete example.com
```

3) Update a site (commonly used for SSL) ?

Command

```
ee site update [<site-name>] [--ssl=<ssl>] [--wildcard]
```

Options

1. `[<site-name>]`: Site to update (if omitted, behavior depends on EE context/version).
2. `--ssl=<ssl>`: Enable SSL on an existing site.
3. `--wildcard`: Enable wildcard SSL.

Examples

1. Add Let's Encrypt SSL:

```
ee site update example.com --ssl=le
```

2. Add wildcard Let's Encrypt SSL:

```
ee site update example.com --ssl=le --wildcard
```

3. Add self-signed SSL:

```
ee site update example.com --ssl=self
```

4) Enable / Disable a site ??

Enable (starts containers if stopped)

```
ee site enable [<site-name>] [--force] [--verify]
```

Options

1. `--force`: Force execution.
2. `--verify`: Verify dependent global services (slower, but safer).

Examples

1. Enable site:

```
ee site enable example.com
```

2. Enable with dependency verification:

```
ee site enable example.com --verify
```

3. Force enable:

```
ee site enable example.com --force
```

Disable (stops and removes site containers)

```
ee site disable [<site-name>]
```

Example

```
ee site disable example.com
```

5) SSL verify/renew ?

Command

```
ee site ssl <site-name> [--force]
```

Options

1. `--force`: Force renewal.

Example

```
ee site ssl example.com --force
```

6) Site info ??

Command

```
ee site info [<site-name>]
```

Example

```
ee site info example.com
```

7) List sites ?

Command

```
ee site list [--enabled] [--disabled] [--format=<format>]
```

Options

1. `--enabled`: Only enabled sites.
2. `--disabled`: Only disabled sites.
3. `--format=<format>`: Output format.
 1. `table` (default)
 2. `csv`
 3. `yaml`
 4. `json`
 5. `count`
 6. `text`

Examples

1. List all:

```
ee site list
```

2. List enabled:

```
ee site list --enabled
```

3. List disabled:

```
ee site list --disabled
```

4. JSON output:

```
ee site list --format=json
```

5. Count:

```
ee site list --format=count
```

8) Reload / Restart (pattern) ?

Your docs show the *same argument pattern* for several actions:

```
ee site reload --type=<type> <site-name> [--ssl=<value>] [--wildcard] [--skip-status-check]
ee site restart --type=<type> <site-name> [--ssl=<value>] [--wildcard] [--skip-status-check]
```

Notes

- `--type=<type>`: Site type (`html`, `wp`, `php`, etc.).
- `--ssl=<value>` and `--wildcard` follow the same meaning as create/update.
- These commands are typically used for *re-provisioning / restarting* the site stack.

Example

```
ee site restart --type=html example.com
```

9) Share a site via ngrok ?

Command

```
ee site share <site-name> [--disable] [--refresh] [--token=<token>]
```

Options

1. `--disable`: Take the shared link down.
2. `--refresh`: Refresh if share link expired.
3. `--token=<token>`: ngrok auth token.

Examples

1. Share:

```
ee site share example.com
```

2. Refresh:

```
ee site share example.com --refresh
```

3. Disable:

```
ee site share example.com --disable
```

10) Clear cache (page/object) ?

Command

```
ee site clean [<site-name>] [--page] [--object]
```

Options

1. `--page`: Clear page cache.
2. `--object`: Clear object cache.

Examples

1. Clear both (no flags):

```
ee site clean example.com
```

2. Clear object cache:

```
ee site clean example.com --object
```

3. Clear page cache:

```
ee site clean example.com --page
```

WordPress Sites (`--type=wp`) ?

1) Create a WordPress site

Command

```
ee site create --type=wp <site-name> [--cache] [--vip] [--mu=<subdir>] [--mu=<subdom>] [--title=<title>] \  
  [--admin-user=<admin-user>] [--admin-pass=<admin-pass>] [--admin-email=<admin-email>] \  
  [--local-db] [--with-local-redis] [--php=<php-version>] \  
  [--dbname=<dbname>] [--dbuser=<dbuser>] [--dbpass=<dbpass>] [--dbhost=<dbhost>] \  
  [--dbprefix=<dbprefix>] [--dbcharset=<dbcharset>] [--dbcollate=<dbcollate>] \  
  [--skip-check] [--version=<version>] [--skip-content] [--skip-install] [--skip-status-check] \  
  [--ssl=<value>] [--wildcard] [--yes] [--force]
```

Most-used options

1. Core install

1. `--title=<title>`
2. `--admin-user=<admin-user>`
3. `--admin-pass=<admin-pass>`
4. `--admin-email=<admin-email>`

2. SSL

1. `--ssl=le` / `--ssl=self`
2. `--wildcard`

3. Caching / Redis

1. `--cache`: Enable Redis cache for WP.
2. `--with-local-redis`: Use a *site-local* Redis container.

4. Multisite

1. `--mu=subdir`: Multisite with subdirectories.
2. `--mu=subdom`: Multisite with subdomains.

5. Database placement

1. `--local-db`: Separate DB container for this site instead of global DB.
2. Remote DB:
 1. `--dbhost=<dbhost>`
 2. `--dbuser=<dbuser>`
 3. `--dbpass=<dbpass>`
 4. `--skip-check` (skip DB connection check)
 5. `--force` (reset remote DB if not empty)

6. WP version/content

1. `--version=<version>`: `latest`, `nightly`, or a version number.

2. `--skip-content`: Don't download default themes/plugins.
3. `--skip-install`: Skip `wp core install`.

PHP version

1. `--php=<php-version>` (as per your docs)
 1. `5.6`
 2. `7.2`
 3. `latest` (default)

Examples

1. Basic WP site:

```
ee site create example.com --type=wp
```

2. Multisite (subdir):

```
ee site create example.com --type=wp --mu=subdir
```

3. Multisite (subdom):

```
ee site create example.com --type=wp --mu=subdom
```

4. WP + Let's Encrypt:

```
ee site create example.com --type=wp --ssl=le
```

5. WP + wildcard SSL:

```
ee site create example.com --type=wp --ssl=le --wildcard
```

6. WP + self-signed SSL:

```
ee site create example.com --type=wp --ssl=self
```

7. WP with remote DB:

```
ee site create example.com --type=wp --dbhost=localhost --dbuser=username --  
dbpass=password
```

8. WP with custom title/admin (as in your docs; note: `--locale` appears in example though not listed in options):

```
ee site create example.com --type=wp --title=easyengine --locale=nl_NL \  
--admin-email=easyengine@example.com --admin-user=easyengine --admin-  
pass=easyengine
```

2) WP reload / restart (pattern) ?

These mirror the WP create flags:

```
ee site reload --type=wp <site-name> [same flags as create...]  
ee site restart --type=wp <site-name> [same flags as create...]
```

Example

```
ee site restart --type=wp example.com
```

Cron Management (ee cron) ??

1) Overview / help entry

```
ee cron
```

2) Create a cron job

Command

```
ee cron create [<site-name>] --command=<command> --schedule=<schedule> [--user=<user>]
```

Targets

1. Site cron: pass `<site-name>` (e.g., `example.com`)
2. Host cron: use `host` as site-name target (per your examples)

Schedule formats

1. Standard Linux cron: e.g. `* * * * *`
2. Shortcut macros:
 1. `@yearly` / `@annually` → `0 0 1 1 *`
 2. `@monthly` → `0 0 1 * *`
 3. `@weekly` → `0 0 * * 0`
 4. `@daily` / `@midnight` → `0 0 * * *`
 5. `@hourly` → `0 * * * *`
3. Fixed interval:
 1. `@every <duration>`

1. duration units: `h`, `m`, `s`
2. examples: `10m`, `1h`, `1h10m2s`

Examples

1. Run WP cron due events every 10 minutes (site):

```
ee cron create example.com --command='wp cron event run --due-now' --schedule='@every 10m'
```

2. Run every minute with Linux cron format:

```
ee cron create example.com --command='wp cron event run --due-now' --schedule='* * * * *'
```

3. Run every minute as `www-data`:

```
ee cron create example.com --command='wp cron event run --due-now' --schedule='* * * * *' --user=www-data
```

4. Add cron on the host machine:

```
ee cron create host --command='wp cron event run --due-now' --schedule='@every 10m'
```

5. Weekly cron on host:

```
ee cron create host --command='wp media regenerate --yes' --schedule='@weekly'
```

3) List cron jobs

Command

```
ee cron list [<site-name>] [--all]
```

Examples

1. List cron jobs (default view):

```
ee cron list
```

2. List cron jobs for one site:

```
ee cron list example.com
```

3. View all cron jobs:

```
ee cron list --all
```

4) Update a cron job

Command

```
ee cron update <id> [--site=<site>] [--command=<command>] [--schedule=<schedule>] [--user=<user>]
```

Examples

1. Change site target:

```
ee cron update 1 --site='example1.com'
```

2. Change command:

```
ee cron update 1 --command='wp cron event run --due-now'
```

3. Change command + user:

```
ee cron update 1 --command='wp cron event run --due-now' --user=root
```

4. Change schedule:

```
ee cron update 1 --schedule='@every 1m'
```

5) Run a cron job now ??

Command

```
ee cron run-now <cron-id>
```

Example

```
ee cron run-now 1
```

6) Delete a cron job ??

Command

```
ee cron delete <cron-id>
```

Example

```
ee cron delete 1
```

Shell / Exec Into Containers (`ee shell`) ?

Open a shell (interactive) or run a command (non-interactive)

Command

```
ee shell [<site-name>] [--user=<user>] [--service=<service>] [--command=<command>] [--skip-tty]
```

Options

1. `[<site-name>]`: Site name to enter.
2. `--user=<user>`: Exec as a specific user (e.g., `root`).
3. `--service=<service>`: Which service/container to exec into.
 1. default: `php`
4. `--command=<command>`: Run a single command non-interactively.
5. `--skip-tty`: Don't allocate a TTY (useful in CI).

Examples

1. Open default shell (PHP service):

```
ee shell example.com
```

2. Open shell as root:

```
ee shell example.com --user=root
```

3. Open shell into nginx:

```
ee shell example.com --service=nginx
```

4. Run a one-shot command in nginx:

```
ee shell example.com --service=nginx --command='nginx -t && nginx -s reload'
```

“Fast Lookup” Index ?

1. **Create HTML site:** `ee site create --type=html example.com`
2. **Create WP site:** `ee site create --type=wp example.com`
3. **Add SSL:** `ee site update example.com --ssl=le`
4. **Force renew SSL:** `ee site ssl example.com --force`
5. **Enable / Disable:** `ee site enable example.com` / `ee site disable example.com`
6. **Info:** `ee site info example.com`
7. **List sites:** `ee site list --format=json`
8. **Share via ngrok:** `ee site share example.com`
9. **Clear cache:** `ee site clean example.com --page` (or `--object`)
10. **Cron add:** `ee cron create example.com --command='...' --schedule='@every 10m'`
11. **Cron list:** `ee cron list --all`
12. **Shell:** `ee shell example.com --service=php`

If you tell me which EasyEngine major version you're on (v4/v5+) and which site types you actually use (html/php/wp), I can tailor this into a *one-page* condensed cheat sheet plus a *troubleshooting mini-section* (common errors + exact commands to diagnose).

Terminal “GUI” for EasyEngine (Debian) ???

Not tested yet.

“ I've installed EasyEngine on my Debian root server. Could you write me a script that displays a GUI in the terminal where I can do everything? So that it runs the ee commands in the background but displays everything graphically for me. For example, I'd like to be able to view, modify, and create all my websites, etc.

You can absolutely build a *terminal UI (TUI)* that **wraps** `ee`, runs commands “in the background,” and lets you **list / create / update / delete / enable / disable** sites, view info, manage SSL, etc.

The most maintainable approach on Debian is:

- **Python** + `Textual` (modern TUI framework, mouse/keyboard, panels, logs) □
- Spawn `ee` **subprocesses**, stream output to a “log pane,” and show progress/status.

Below is a *working starter app* you can extend into “do everything.” It already supports:

1. **List sites** (and refresh)
2. **Show site info**
3. **Enable / Disable**
4. **Delete** (with confirmation)
5. **Create** (HTML/WP, optional SSL)
6. **Update SSL** (set/renew)

“ ⚠ **Security note:** This kind of wrapper typically needs to run as **root** (or a user with permission to run Docker + manage `/opt/easyengine`). Be careful about storing admin passwords (for WP create) and about allowing arbitrary command execution.

1) Install prerequisites (Debian)

```
apt-get update
apt-get install -y python3 python3-pip
python3 -m pip install --upgrade pip
python3 -m pip install textual rich
```

Verify EasyEngine works:

```
ee --version
ee site list --format=json
```

2) Save the TUI script

Create a file: `ee-tui.py`

```
#!/usr/bin/env python3
import asyncio
import json
import shlex
from dataclasses import dataclass
from typing import Any, Dict, List, Optional

from textual.app import App, ComposeResult
from textual.containers import Horizontal, Vertical
from textual.message import Message
from textual.reactive import reactive
from textual.widgets import (
    Button,
    DataTable,
    Footer,
    Header,
    Input,
    Label,
    Log,
    Select,
    Static,
)

EE_BIN = "ee"
```

```

def safe_site_name(name: str) -> str:
    name = (name or "").strip()
    # Minimal guardrails; EE will validate further.
    if not name or " " in name or "/" in name:
        raise ValueError("Invalid site name.")
    return name

async def run_ee(args: List[str], timeout: Optional[int] = None) -> Dict[str, Any]:
    """
    Run ee command and capture stdout/stderr. Returns dict with rc/stdout/stderr.
    """
    cmd = [EE_BIN] + args
    proc = await asyncio.create_subprocess_exec(
        *cmd,
        stdout=asyncio.subprocess.PIPE,
        stderr=asyncio.subprocess.PIPE,
    )
    try:
        stdout, stderr = await asyncio.wait_for(proc.communicate(), timeout=timeout)
    except asyncio.TimeoutError:
        proc.kill()
        raise
    return {
        "rc": proc.returncode,
        "stdout": (stdout or b "").decode(errors="replace"),
        "stderr": (stderr or b "").decode(errors="replace"),
        "cmd": " ".join(shlex.quote(c) for c in cmd),
    }

async def stream_ee(args: List[str], on_line, on_done) -> None:
    """
    Run ee and stream output lines to callback.
    """
    cmd = [EE_BIN] + args
    proc = await asyncio.create_subprocess_exec(
        *cmd,

```

```

        stdout=asyncio.subprocess.PIPE,
        stderr=asyncio.subprocess.STDOUT,
    )
    await on_line(f"$ {' '.join(shlex.quote(c) for c in cmd)}\n")
    assert proc.stdout is not None
    async for raw in proc.stdout:
        await on_line(raw.decode(errors="replace"))
    rc = await proc.wait()
    await on_done(rc)

```

```
@dataclass
```

```
class SiteRow:
```

```

    name: str
    type: str
    enabled: str
    ssl: str

```

```
class StatusBar(Static):
```

```

    text = reactive("Ready.")

    def render(self) -> str:
        return f"**Status:** {self.text}"

```

```
class EETui(App):
```

```

    CSS = """
    Screen { layout: vertical; }
    #main { height: 1fr; }
    #left { width: 1fr; }
    #right { width: 1fr; }
    #actions Button { margin: 1 1; }
    #form Input, #form Select { margin: 1 0; }
    #log { height: 1fr; border: round $surface; }
    #table { height: 1fr; border: round $surface; }
    #info { height: 1fr; border: round $surface; padding: 1; }
    """

```

```
BINDINGS = [
```

```

    ("q", "quit", "Quit"),
    ("r", "refresh", "Refresh Sites"),
]

# UI state
sites: List[SiteRow] = []
selected_site: Optional[str] = reactive(None)

def compose(self) -> ComposeResult:
    yield Header(show_clock=True)
    with Horizontal(id="main"):
        with Vertical(id="left"):
            yield Label("Websites ☐☐", id="title_left")
            table = DataTable(id="table", zebra_stripes=True)
            table.add_columns("Name", "Type", "Enabled", "SSL")
            yield table
            yield StatusBar(id="status")
        with Vertical(id="right"):
            yield Label("Actions ☐☐", id="title_right")
            with Horizontal(id="actions"):
                yield Button("Refresh", id="btn_refresh", variant="primary")
                yield Button("Info", id="btn_info")
                yield Button("Enable", id="btn_enable")
                yield Button("Disable", id="btn_disable")
                yield Button("Delete", id="btn_delete", variant="error")

            yield Label("Create / Update ☐")
            with Vertical(id="form"):
                yield Input(placeholder="site name (e.g. example.com)", id="in_name")
                yield Select(
                    options=[
                        ("html", "html"),
                        ("wp", "wp"),
                    ],
                    prompt="site type",
                    id="sel_type",
                )
                yield Select(
                    options=[
                        ("none", "none"),

```

```

        ("le", "le (Let's Encrypt)",
        ("self", "self (self-signed)",
        ],
        prompt="SSL (optional)",
        id="sel_ssl",
    )
    with Horizontal():
        yield Button("Create Site", id="btn_create", variant="success")
        yield Button("Set SSL", id="btn_setssl")

    yield Label("Output ☐☐")
    yield Log(id="log", highlight=True, auto_scroll=True)

    yield Label("Site Info ☐☐")
    yield Static("", id="info")

    yield Footer()

async def on_mount(self) -> None:
    await self.refresh_sites()

def _status(self, msg: str) -> None:
    self.query_one(StatusBar).text = msg

def _log(self, text: str) -> None:
    self.query_one(Log).write(text.rstrip("\n"))

async def refresh_sites(self) -> None:
    self._status("Refreshing sites...")
    self._log("Refreshing site list...")
    res = await run_ee(["site", "list", "--format=json"])
    if res["rc"] != 0:
        self._status("Failed to refresh.")
        self._log(res["stderr"] or res["stdout"])
        return

    try:
        data = json.loads(res["stdout"] or "[]")
    except json.JSONDecodeError:
        self._status("Failed to parse site list.")

```

```

        self._log(res["stdout"])
        return

sites: List[SiteRow] = []
for item in data:
    # Fields vary slightly by EE versions; keep it tolerant.
    name = str(item.get("name") or item.get("site") or "")
    typ = str(item.get("type") or "")
    enabled = str(item.get("enabled") or item.get("status") or "")
    ssl = str(item.get("ssl") or item.get("https") or "")
    if name:
        sites.append(SiteRow(name=name, type=typ, enabled=enabled, ssl=ssl))

self.sites = sorted(sites, key=lambda s: s.name)

table = self.query_one(DataTable)
table.clear()
for s in self.sites:
    table.add_row(s.name, s.type, s.enabled, s.ssl, key=s.name)

self._status(f"Loaded {len(self.sites)} site(s).")

async def action_refresh(self) -> None:
    await self.refresh_sites()

async def on_data_table_row_selected(self, event: DataTable.RowSelected) -> None:
    self.selected_site = str(event.row_key)

async def _stream_command(self, args: List[str]) -> int:
    self.query_one(Log).write("") # spacer
    self._status("Running command...")
    rc_holder = {"rc": 1}

    async def on_line(line: str):
        self._log(line)

    async def on_done(rc: int):
        rc_holder["rc"] = rc
        self._status(f"Done (rc={rc}).")

```

```

await stream_ee(args, on_line, on_done)
return rc_holder["rc"]

def _get_selected(self) -> str:
    if not self.selected_site:
        raise ValueError("No site selected.")
    return safe_site_name(self.selected_site)

async def on_button_pressed(self, event: Button.Pressed) -> None:
    bid = event.button.id

    try:
        if bid == "btn_refresh":
            await self.refresh_sites()

        elif bid == "btn_info":
            site = self._get_selected()
            rc = await self._stream_command(["site", "info", site])
            if rc == 0:
                res = await run_ee(["site", "info", site])
                self.query_one(Static, id="info").update(
                    f"***{site}**\n\n```\n{(res['stdout'] or '').strip()}\n```"
                )

        elif bid == "btn_enable":
            site = self._get_selected()
            rc = await self._stream_command(["site", "enable", site])
            if rc == 0:
                await self.refresh_sites()

        elif bid == "btn_disable":
            site = self._get_selected()
            rc = await self._stream_command(["site", "disable", site])
            if rc == 0:
                await self.refresh_sites()

        elif bid == "btn_delete":
            site = self._get_selected()
            # Lightweight "confirmation" via status prompt style:
            self._status(f"Deleting {site}... (no undo)")

```

```

rc = await self._stream_command(["site", "delete", site, "--yes"])
if rc == 0:
    self.selected_site = None
    self.query_one(Static, id="info").update("")
    await self.refresh_sites()

elif bid == "btn_create":
    name = safe_site_name(self.query_one(Input, id="in_name").value)
    site_type = self.query_one(Select, id="sel_type").value or "html"
    ssl = self.query_one(Select, id="sel_ssl").value or "none"

    args = ["site", "create", name, f"--type={site_type}"]
    if ssl != "none":
        args.append(f"--ssl={ssl}")

    # For WP, EE may prompt unless defaults are allowed; you can add fields later.
    rc = await self._stream_command(args)
    if rc == 0:
        await self.refresh_sites()

elif bid == "btn_setssl":
    site = self._get_selected()
    ssl = self.query_one(Select, id="sel_ssl").value or "none"
    if ssl == "none":
        raise ValueError("Select an SSL option (le/self) first.")
    rc = await self._stream_command(["site", "update", site, f"--ssl={ssl}"])
    if rc == 0:
        await self.refresh_sites()

except Exception as e:
    self._status(f"Error: {e}")
    self._log(f"Error: {e}")

if __name__ == "__main__":
    EETui().run()

```

Make it executable:

```
chmod +x ee-tui.py
```

Run it:

```
sudo ./ee-tui.py
```

3) What you'll likely want to add next ?

1. **WordPress “create” form fields** (title/admin user/email/pass, multisite, cache/redis, php version).
 2. **Site clean / cache controls**
 - `ee site clean <site> --page / --object`
 3. **SSL renew button**
 - `ee site ssl <site> --force`
 4. **Logs viewer / service status**
 - Shell into containers: `ee shell <site> --service=nginx|php`
 5. **Cron management panel**
 - `ee cron list/create/update/delete`
-

4) A couple of questions so I tailor it to “everything” ?

1. Which EasyEngine major version are you on (**v4** or **v5+**)? (`ee --version`)
2. Which site types do you use besides `html/wp` (e.g., `php`, `proxy`, etc.)?
3. Do you want the app to support **remote DB**, **WP multisite**, and **Redis cache toggles** from the UI?

If you answer those, I can expand this into a fuller “control panel” TUI with tabs (Sites / WP / SSL / Cron / Shell), better confirmations, and safer privilege handling.

c1) Orientation: What EasyEngine is (and why it changes your workflow) ?

c1) Orientation: What EasyEngine is (and why it changes your workflow) ☐☐

c1.1 — What EasyEngine CLI is (and why it's worth switching) ?

What EasyEngine CLI does

EasyEngine (`ee`) is a **server-side CLI** that helps you **provision and operate WordPress and other web stacks** using **containers** (commonly Docker-based). Instead of clicking around in a shared-hosting control panel, you manage sites with **repeatable commands**, which makes your workflow:

1. **Faster**
 - Create, update, and operate sites in seconds/minutes.
2. **More consistent**
 - The same commands produce the same setup every time (great for staging/prod parity).
3. **More operationally powerful**
 - You get direct access to services like **Nginx, PHP, DB, Redis**, and the ability to automate routines (backups, cron, cache clears, etc.).

How it “thinks”: sites as managed units

With EasyEngine, a **site** (like `example.com`) becomes a **managed unit** you can operate with subcommands such as:

1. **Lifecycle**
 - Create, enable/disable, update, delete
2. **Operations**
 - Reload/restart services, clear caches (`ee site clean`)
3. **Access**
 - Shell into the site's containers (`ee shell`)
4. **Publishing & sharing**
 - Temporarily expose a local/secured site via a share link (`ee site share`) ☐☐
5. **Scheduling**
 - Manage cron jobs with `ee cron` ☐

This aligns strongly with real DevOps-style workflows: **declare intent** → **run command** → **verify state**.

What changes vs shared hosting (the practical mindset shift)

You're moving from “**panel-driven hosting**” to “**infrastructure you control**”:

1. **You manage the server**
 - Updates, firewall, SSH keys, resources, backups become *your* responsibility.
 2. **You gain performance & control**
 - Tuning PHP/Nginx, adding Redis, isolating sites, debugging via logs/shell access.
 3. **You automate more**
 - Commands are scriptable; cron becomes explicit; troubleshooting becomes faster.
-

Core tools you'll use daily (from your excerpt)

Here's how the parts in your provided docs fit together:

1. `ee site ...`
 - Day-to-day site operations
 - Examples from excerpt:
 - `ee site clean` → clears page/object cache
 - `ee site share` → share a site via an online link (with `--refresh`, `--disable`, `--token=...`)
 2. `ee shell ...`
 - Your “daily driver” to run WP-CLI, inspect configs, read logs inside the right container context
 3. `ee cron ...`
 - Create/list/update/delete/run scheduled tasks cleanly
-

Outcomes you can expect after this course ?

By the time you complete the chapters, you should be able to:

1. Provision WordPress sites confidently (including PHP/DB choices)
 2. Operate and debug sites via `ee shell`
 3. Manage caching and routine maintenance (`ee site clean`, reload/restart)
 4. Create reliable scheduled tasks with `ee cron`
 5. Use share links safely and intentionally (`ee site share`)
-

Next: pick c1.2 or tell me your goal ?

If you want a natural continuation, I suggest:

1. **c1.2 — Installation & prerequisites** (server requirements, Docker, DNS, ports, safety basics), or
2. **c2.4 — Create a WordPress site end-to-end** (fastest path to “I’m productive”), or
3. Tell me: *Do you plan to run one server for multiple client sites, or one server per project?*