ChatGPT

# PMOVES.AI Provisioning Bundle Overview

The **PMOVES.AI Provisioning Bundle** is a self-contained package that streamlines setting up the entire PMOVES multi-agent stack (Agent Zero, Archon, Hi-RAG gateways, etc.) along with key external services [1]. It includes updated scripts, configuration templates, and Docker Compose overrides to **onboard new users automatically** and launch a full PMOVES environment with minimal manual steps. This bundle integrates with the PMOVES CLI tools to walk users through initial configuration ("**first-run wizard**") and perform environment diagnostics (pre-flight checks and smoke tests).

## Bundle Contents and Structure

The ZIP bundle is organized as a drop-in addon for the `PMOVES.AI` repository. Once unzipped into the repository, it adds or updates the following files and directories (relative to the repo root):

- **Provisioning Scripts:**
- `scripts/install/wizard.sh` – Interactive first-run setup wizard (Bash) [2] [3].
- `scripts/install/wizard.ps1` – First-run wizard for Windows PowerShell (analogous to the Bash wizard).
- `scripts/proxmox/pmoves-bootstrap.sh` – Automated bootstrap script for Proxmox-based deployments (helps provision PMOVES on Proxmox VMs/nodes).
- **Configuration Templates:**
- `.env.example` – Sample environment file with new external-service flags and defaults (e.g. `EXTERNAL_*` toggles) [4].
- `env.shared.example` – Sample shared secrets file containing placeholders for API keys, tokens, and credentials for all services (Supabase, Wger, Firefly III, Discord, Jellyfin, etc.) [5] [6]. Users copy this to `pmoves/env.shared` and fill in real values.
- **Docker Compose overrides:** `docker-compose.gpu.yml` for GPU-accelerated services (Hi-RAG GPU gateway) and any necessary compose file additions for external integrations (e.g. Jellyfin AI transcoding profile). These ensure that running `make up-gpu` adds GPU variants where available [7] [8].
- **Makefile patches:** Updates to the main `Makefile` introducing new targets and profiles for external services and diagnostics (described below). These are provided as unified diff patches (e.g. `patches/Makefile.diff`) which can be applied to your repo [9], or you can manually merge the changes since the new files are included in entirety. Key modifications include targets like `up-external-wger`, `up-external-firefly`, `up-external-jellyfin`, `discord-ping`, etc., and support for backup/restore and GPU profiles.
- **Documentation updates:** Patches to docs (in `patches/docs_*.diff`) that update the Quickstart and tooling guides with instructions for the new services (Firefly III, Wger, Jellyfin) and the use of the provisioning wizard and smoke tests [10]. A top-level `README.md` is included in the bundle summarizing usage (much like this answer section), so new users have one clear guide.

Below is the **directory structure** of the bundle content:

```
pmoves-provisioning-bundle/   (root of extracted bundle)
├── README.md                 # Guide for setup and usage of this bundle
├── .env.example              # Updated sample env (core settings & external
flags)
├── env.shared.example        # Sample shared secrets (API keys, tokens,
credentials)
├── docker-compose.gpu.yml    # New compose override for GPU support
├── patches/                  # Patches for integrating into PMOVES.AI repo
│   ├── Makefile.diff
│   ├── env.example.diff
│   ├── docker-compose.yml.diff
│   ├── docs_LOCAL_DEV.md.diff
│   ├── docs_MAKE_TARGETS.md.diff
│   └── docs_SMOKETESTS.md.diff
└── scripts/
    ├── install/
    │   ├── wizard.sh         # First-run wizard (Unix)
    │   └── wizard.ps1        # First-run wizard (Windows)
    └── proxmox/
        └── pmoves-bootstrap.sh  # Proxmox provisioning helper
```

*(All scripts in `scripts/` should be made executable after extraction.)*

## Configuration & Onboarding Setup

**1. Extract and Configure:** To use the bundle, unzip it at the root of your PMOVES.AI repository. This will add the new scripts and merge patch updates. If using Git, it's recommended to create a new branch and apply the provided diffs (e.g. `git apply patches/Makefile.diff` etc.) [9], then commit. Next, copy the sample env files into place: - `pmoves/.env.example` → `pmoves/.env` (or `.env.local`)
- `pmoves/env.shared.example` → `pmoves/env.shared`
The `.env` file holds configuration toggles and local environment settings, while `env.shared` holds **credentials and API keys** used by services. Fill in all required values in `env.shared` (database URLs, Supabase keys, API tokens, secrets, etc.) before proceeding. For example, provide your Supabase keys, a Wger API token, Firefly III personal token, Discord webhook URL, and any OAuth keys for external providers as needed.

> **External Service Tokens:** If you haven't done so, generate the API tokens for Wger and Firefly III via their web UIs (explained below) and update `env.shared`. The file has placeholders for `WGER_API_TOKEN`, `FIREFLY_ACCESS_TOKEN`, `DISCORD_WEBHOOK_URL`, `JELLYFIN_API_KEY`, etc., which must be filled in for full functionality [5] [6].

**2. Run the Onboarding Wizard:** The bundle provides an interactive wizard to guide initial setup. Launch it from the repository root: - **Unix/macOS:** `bash pmoves/scripts/install/wizard.sh`
- **Windows:** `pwsh -File pmoves/scripts/install/wizard.ps1`

The wizard will prompt for your desired setup options and credentials: - *Stack mode:* choose **Full** (all services including n8n automations, YouTube processing, etc.) or **Minimal** (core PMOVES services only, excluding heavy extras).
- *External services:* indicate (y/N) if you are using external instances for Supabase, Neo4j, Meilisearch, or Qdrant. For each selection, the wizard will set the corresponding `EXTERNAL_*` flag in your `.env` file to `true` or `false` [11]. (If `true`, PMOVES will not launch the local container for that service, expecting you to supply an external endpoint in `env.shared`.)
- *GPU acceleration:* choose whether to enable the GPU profile. If "yes", the wizard will later invoke the stack with GPU-enabled compose overlays (bringing up GPU-accelerated Hi-RAG gateways and Jellyfin transcoding support) [12].
- *Discord webhook:* optionally provide a Discord Webhook URL if you want the system to send notifications (e.g. when publishing content) to a Discord channel. If you input a URL, the wizard writes it into your env file (`DISCORD_WEBHOOK_URL`) [3]. You can skip this for now by leaving it blank – you can always add it later in `env.shared`.

After gathering inputs, the wizard auto-applies the configuration: it updates your `.env` with the external flags and any provided values, then launches the PMOVES stack. Specifically, it runs `make up` (or `make up-gpu` if GPU mode) to start core containers, and if "Full" mode was selected, it additionally starts the n8n workflow engine, YouTube pipeline, and ComfyUI services (via `make up-n8n`, `make up-yt`, `make up-comfy`) [13]. Finally, it runs a **flight-check** (`make flight-check`) to perform basic environment diagnostics and outputs the results [14]. At the end of the wizard, your local PMOVES environment should be up and running.

**3. PMOVES CLI Integration (Alternative):** If you prefer command-line steps or need to automate provisioning, the PMOVES CLI can accomplish similar tasks: - Run `make bootstrap` (or `python -m pmoves.scripts.bootstrap_env`) to interactively bootstrap environment variables and generate the env files from a manifest. This captures things like Supabase service URLs, keys, and generates any needed secrets (e.g. JWT tokens) [15]. It also pulls external service credentials (Wger, Firefly, etc.) that you input. After running this, you'll have a populated `env.shared`.
- Then run the CLI's provisioning command to stage the bundle: `python -m pmoves.tools.mini_cli bootstrap` (or equivalently `pmoves mini-cli bootstrap`). This copies the provisioning bundle into `CATACLYSM_STUDIOS_INC/PMOVES-PROVISIONS` directory [16] [17]. From there you can run the included `install/wizard` script as described above. *(These steps are essentially what the one-step* `wizard.sh` *automates, but they are available if you need non-interactive or partial setup.)*

## Launching Services and Updated Makefile Targets

With the new bundle applied, the **Makefile** now has convenient targets to manage the external integration services and to perform common tasks:

- **Core Stack:** `make up` brings up the core PMOVES data plane and workers (Postgres, Qdrant, Meili, MinIO, Hi-RAG gateways, LangExtract, etc.) using Docker Compose [7]. After configuration, simply running `make up` (or using the wizard as above) will launch the full stack. Use `make down` to stop containers, and `make clean` to tear down and remove volumes if needed (for a fresh reset).
- **External Services (Firefly III, Wger, Open Notebook, Jellyfin):** You can launch these with dedicated targets that attach them to PMOVES's network:

- `make up-external-wger` – Start the **Wger** health tracking service (Django app plus Nginx proxy) on the shared `cataclysm-net` Docker network [18]. This will spin up containers for Wger and its static file proxy, and then automatically run the branding script to customize the instance for PMOVES (site name, admin user defaults, etc.) [18] [19].
- `make up-external-firefly` – Start the **Firefly III** personal finance service on `cataclysm-net` (the container will map to host port **8082** by default, see below) [20].
- `make up-external-on` – Start **Open Notebook** (research notebook UI) using the external Docker image on `cataclysm-net` [21].
- `make up-external-jellyfin` – Start **Jellyfin** media server on `cataclysm-net` (container mapping to host port 8096) [22].
- `make up-external` – (All-in-one) Bring up **all** the above external services in one command [23]. This ensures the Docker networks (`cataclysm-net` and the internal `pmoves-net` for Supabase) exist and then launches Wger, Firefly, Open Notebook, and Jellyfin together.

- To shut down external services, use `make down-external` [24].

- **Integrated Automations:** The Makefile also provides targets to run the **integration workflows** (n8n and flows watcher) alongside external services if you prefer an integrated compose stack:

- `make integrations-up-core` – Launch the n8n workflow automation service by itself (on the `cataclysm-net` network) [25].
- `make integrations-up-wger` / `integrations-up-firefly` – Launch n8n plus the Wger or Firefly service *within a single compose project* for integration testing [26]. (In this mode, Wger and Firefly run with ephemeral data as part of the "integrations" Docker Compose, useful for development or CI.)
- `make integrations-up-all` – Launch n8n with both Wger and Firefly, plus a flows-watcher service, in one go [27]. The flows-watcher automatically imports any JSON flow definitions (found under `pmoves/n8n/flows/`) into n8n. This is handy for setting up the health and finance data sync flows after everything is up.

- After making changes or when tearing down, `make integrations-down` stops the integration stack and removes volumes [28]. There are also helper targets like `integrations-import-flows` to manually import flows via the n8n API and `integrations-logs` to tail the n8n logs [29].

- **Discord Notifications:** PMOVES now includes a **Discord publisher** microservice that posts messages to your Discord webhook (if configured) for certain events (like completed tasks or alerts). This service (`publisher-discord`) is automatically started as part of the agents stack (brought up with Agent Zero, Archon, etc.) [30]. The Makefile offers:

- `make discord-ping` – Send a test message to your configured Discord webhook to verify connectivity. You can set a custom message via the `MSG` environment variable. For example: `make discord-ping MSG="Hello from PMOVES"` will POST a message to the `$DISCORD_WEBHOOK_URL` you set [31]. (If the webhook URL is not set, the command will remind you to configure it before proceeding [32].)
- `make discord-smoke` – Run a slightly more thorough health check of the Discord publishing pipeline. It will check the health endpoint of the Discord publisher service and attempt a sample publish call, ensuring an `{ ok: true } response` [33].

- The environment variables `DISCORD_USERNAME` and `DISCORD_AVATAR_URL` in `env.shared` allow you to customize the bot name and avatar for messages [5] (defaults are provided, e.g. "Cataclysm Publisher" with a PMOVES avatar).

- **GPU and Hardware Profiles:**

- `make up-gpu` – Launch the stack with GPU-supported services (this is equivalent to using `docker-compose.gpu.yml` in conjunction with the default compose) [34]. It enables GPU acceleration for components like the Hi-RAG retrieval model and any AI media processing.
- `make up-jellyfin-ai` – Launch the optional **Jellyfin AI** stack (for hardware-accelerated media analysis in Jellyfin) [35]. This uses a compose overlay for Jellyfin's FFmpeg with VAAPI/NVENC support. (By default Jellyfin runs in software transcoding mode; enabling this profile offloads transcoding to GPU and also starts the Jellyfin AI companion services such as the audio analyzer with HDR support [36].)

- The bundle also includes sample hardware profile YAMLs (under `pmoves/config/profiles/`) for different setups, and a CLI subcommand `pmoves mini-cli profile ...` to manage these. This allows tailoring the compose deployment to specific hardware (e.g., Jetson vs x86 servers), although that is advanced usage.

- **Backup/Restore:**
New Makefile targets for backups have been added: `make backup` will snapshot all core data (Postgres dump, Qdrant snapshot, MinIO bucket mirror, Meilisearch dump) into a timestamped folder [37]. This is useful before applying updates. A corresponding `make restore` prints instructions (since restore may require manual steps) [38]. These help with portability of data in local deployments.

All these targets come with help descriptions – you can run `make help` to see an updated list of available commands. The provisioning bundle's patches update the help output to include the new targets (e.g. smoke tests for Wger, Jellyfin, etc.) [39].

## Default Credentials and Service Access

After launching the stack (either via the wizard or `make up-external` commands), you will have multiple web services running. The bundle provides a summary of default login credentials and URLs for these services:

- **Wger (Health Tracker):** Accessible at `http://localhost:8000` (the Nginx proxy forwards to Wger's Django app) [40] [41]. The default admin login is `admin` **/** `adminadmin` [42]. This password is set by Wger's upstream bootstrap each time it initializes the database. **Important:** It is recommended to change the admin password on first login (via Wger's UI under account settings) [42]. The provisioning wizard's branding step will have already configured the site name, default gym, and admin profile (name/email) according to the `WGER_BRAND_*` values in your env, but the password remains `adminadmin` unless you overrode it via `WGER_BRAND_ADMIN_*` in env. All data in Wger is stored in a local SQLite file (inside the Docker volume); if you ever remove that volume, you'll need to re-run the branding script to reapply these defaults (`make wger-brand-defaults`).

- **Firefly III (Finance Manager):** Accessible at `http://localhost:8082` by default. (The container's internal port is 8080, but we remap it to host **8082** to avoid clashing with Agent Zero's API on 8080 [43]. You can change the host port by editing `FIREFLY_PORT` in your env file if needed.) Firefly III does **not** come with a pre-created user – on first visit, you will be prompted to register an initial account (email and password). Go ahead and create your admin user. Afterwards, to integrate with PMOVES, generate a Personal Access Token in Firefly III (under *Options* → *Profile* → *API keys*) and copy that token into your `env.shared` as `FIREFLY_ACCESS_TOKEN` [43]. PMOVES uses this token for API access (the n8n automation flow will pull transactions data via Firefly's API). If you set `FIREFLY_BASE_URL` and the token correctly, you can run a quick check with `make smoke-firefly` – it will call the Firefly API `/api/v1/about` endpoint to ensure the service is up and the token is working [44].

- **Open Notebook:** Accessible at `http://localhost:8503` (with its API at `http://localhost:5055`). The default credentials for Open Notebook's web UI: **username:** `admin`, **password:** `changeme`. (You can change the password by running `make notebook-set-password PASSWORD=<newpass>` or via the UI). The environment variable `OPEN_NOTEBOOK_API_TOKEN` in env.shared should be set to the token shown in the Open Notebook welcome page, and the URL `OPEN_NOTEBOOK_API_URL` is preset to the internal address `http://cataclysm-open-notebook:5055` (for PMOVES services to communicate with it) [45]. Typically, the wizard or bootstrap process will generate a token for you if it's empty. Once the notebook is running, you can seed it with default AI models by running `make notebook-seed-models` [46].

- **Jellyfin (Media Server):** Accessible at `http://localhost:8096` (default Jellyfin web UI) [41]. On first launch, Jellyfin will present a setup wizard in the browser: you'll create an admin username and password of your choosing (there is no hardcoded default user). Complete the Jellyfin initial setup (it's quick – just language, account, no media libraries need to be added immediately unless you want to). Once done, open **Jellyfin's dashboard** and note your **API Key** (create one under *Dashboard* → *API Keys* if none exists). Put this in your `env.shared` as `JELLYFIN_API_KEY`, and also grab your user's internal ID (the admin user's ID can be found in the URL when editing the profile, or via the API). Set `JELLYFIN_USER_ID` accordingly (this is optional for certain integrations; if unspecified, PMOVES will attempt to find a default user). The bundle already sets `JELLYFIN_URL` (internal URL for services) and `JELLYFIN_PUBLISHED_URL` (the public/base URL, defaulting to `http://localhost:8096` unless you customize it) [47]. It also configures hardware acceleration settings via env (e.g., `JELLYFIN_HWACCEL_MODE`, `JELLYFIN_*_DEVICE` variables) – you can adjust these if you plan to enable transcoding with GPU. By default, Jellyfin is initially empty; PMOVES provides a helper to create the standard media library folders on your host: run `make jellyfin-folders` to create `pmoves/data/jellyfin/Media/{Movies,Music,…}` directories and mount them into Jellyfin [48] [35]. You can drop some sample media files there to test, or integrate with the PMOVES Jellyfin bridge as described in the documentation.

- **PMOVES Core Services:** Most core services (Agent Zero, Archon, etc.) do not have user-facing login interfaces, but a few have admin consoles:

- **Supabase:** If you ran the local Supabase (via `make supa-start` during bootstrap or using the Docker container), the studio is at `http://localhost:54323` with default email `admin@admin.com` (password was shown in the terminal when Supabase started, or use the one

you set in `env.shared`). This is mainly for debugging; the PMOVES stack interacts with Supabase directly via its APIs.
- **N8N (Workflow Automations):** Accessible at `http://localhost:5678` (the editor UI). The first time, you'll create an account (or use the default user if configured). The provisioning doesn't set a default user here for security; just follow the on-screen setup. After login, you should find the PMOVES example workflows imported (if you used `make integrations-import-flows` or the flows-watcher).
- **MinIO:** If running (comes up with core stack), accessible at `http://localhost:9001` (console) with default credentials *minioadmin/minioadmin* unless you changed `MINIO_USER`/ `MINIO_PASSWORD` in env [49] .
- **Agent Zero API:** Port 8080 (if you visit `http://localhost:8080` you get Agent Zero's API root – mainly used by internal services). Hi-RAG Gateway v2 is on 8086 (CPU) and 8087 (GPU, if enabled) for retrieval augmented generation queries.

After setup, you can refer to this summary when logging into each interface. The bundle's README and the PMOVES docs also reiterate these URLs and credentials [41] . It's good practice to change any default passwords on first use and update your `env.shared` if needed.

## Verification and Smoke Tests

To ensure everything is working correctly, the provisioning bundle includes **flight checks** and **smoke test** utilities:

- **Flight Check:** Run `make flight-check` after initial setup (the wizard runs this automatically as well). This checks for common issues: missing dependencies, port conflicts, presence of env files, and basic schema validity. It will output a report of installed CLI tools (Git, Docker, Node, etc.), confirm that required files (like `contracts/topics.json`) are present and valid, and list any ports already in use [50] [51] . For example, it verifies that ports like 6333 (Qdrant), 5432 (Postgres), etc., are free or in use, so you know if any service failed to start due to a conflict [51] . It also does an env synchronization check to make sure your `env.shared` secrets align with any `.env.local` or Supabase config (warns if something critical is missing or looks wrong) [52] [53] . A successful flight-check will end with "Done." and no error messages. If anything is flagged, address it (e.g. kill any process occupying a needed port, or fill in a missing env value) and run `make flight-check` again.

- **Smoke Tests:** A suite of one-command tests is provided to sanity-check each component:

- `make smoke-wger` – Verifies that the Wger UI is reachable and serving static assets [54] [55] . It will print the HTTP status of the Wger homepage and its logo image to ensure both the Django app and Nginx static server are working (expect "Wger smoke passed." if successful).
- `make smoke-firefly` – Checks Firefly III's API as mentioned. Ensure `FIREFLY_ACCESS_TOKEN` is set before running this. It will attempt to fetch the Firefly instance's "about" info via curl and report success or failure.
- `make smoke-archon` – Pings the Archon service's health endpoint (requires Archon to be up, which is started with the agents). This ensures Archon is connected to NATS and Supabase properly [56] .

- `make smoke-langextract` – Tests the LangExtract service by feeding a sample text and ensuring chunks are extracted and stored.
- `make smoke-presign-put` – Tests the MinIO presigned URL upload flow (uploads a dummy file through the presigner service).
- `make smoke-pdf` – Sends a sample PDF through the PDF ingestion pipeline (requires the PDF ingest service running) to ensure the pipeline from upload to Qdrant indexing works.
- `make jellyfin-verify` – Checks that the Jellyfin bridge service can connect to Jellyfin with the given API key and that the Jellyfin server is accessible [35]. This uses the `JELLYFIN_URL` and `JELLYFIN_API_KEY` from env to validate authentication and will log a success message if everything is okay. Make sure you have created an API key and updated the env before running this.
- `make discord-ping` – (As described) sends a test webhook message. Use this to confirm your Discord notifications are set up. There is also `make discord-smoke` which, beyond a basic ping, triggers the Publisher-Discord service's `/publish` endpoint to simulate an internal message – this ensures that the internal service is running and can reach Discord [33].
- **Other tests:** The Makefile help will list more, such as `make mindmap-smoke`, `make smoke-geometry`, etc., which test the CHIT knowledge graph and geometry pipeline parts of PMOVES. These are more related to core AI functions (e.g., ensuring the hi-rag gateways respond to queries). After focusing on the provisioning of external services, you can run these to verify the AI/agent workflow is functioning (they are unchanged by this bundle).

Run `make smoke` to execute a broad set of smoke tests sequentially (this covers many of the above in one go). All tests use non-destructive sample data. If any test fails, the output will indicate which step failed – you can then inspect logs (e.g., `docker compose logs <service>`) for that service to diagnose. The bundle's documentation patch includes a troubleshooting section for common issues (for example, if Wger didn't start, or Firefly's migrations aren't finished, etc. – often just waiting a bit and re-running the smoke test helps, especially on first launch when containers may still be initializing databases).

Finally, you can always rerun `make flight-check` at any time to do a quick environment audit, and use `make ps` to see the status of all containers. The provisioning is designed to be **idempotent** – you can run the wizard or the branding scripts multiple times without harm. For instance, if you realize you needed to change an env variable (say, use a new API token), just update `env.shared` and run the relevant service's up command again or its smoke test to confirm the new setting.

## Conclusion

This provisioning bundle brings together all the pieces needed to get PMOVES.AI and its extended ecosystem up and running quickly. By integrating with the PMOVES CLI and Makefile, it provides an **automated onboarding flow** and one-step commands to launch **Firefly III**, **Wger**, **Jellyfin**, and Discord integrations alongside the core multi-agent platform. After applying the bundle and running the wizard, you should be able to hit `make up` (or `make up-external`) and have a fully configured stack running – from the Supabase backend and MinIO storage to the Agent orchestration layer (Agent Zero, Archon) and all external apps. The default credentials and URLs above give you immediate access to each service's UI, and the included diagnostics help verify every component. This ensures a smooth setup that is **compatible with PMOVES's architecture** (nothing conflicts with existing ports or data flows [57]) and ready to support the PMOVES multi-agent workflows (finance and health data syncing to Supabase, media analysis pipelines, and more).

**Next Steps:** After verification, you can proceed to activate the n8n flows for Wger and Firefly (if you haven't already – the JSON flows are available in `pmoves/n8n/flows/` and can be imported via `make integrations-import-flows` or through the n8n UI). These will start pulling Wger workout logs and Firefly transactions into Supabase, allowing Agent Zero and Archon to reason over your health and finance data. You can also integrate Jellyfin by installing the "Jellyfin for Kodi" plugin or others as needed [58]. With the provisioning bundle, your PMOVES environment should be fully operational — happy exploring!

---

[1] [15] README.md
https://github.com/POWERFULMOVES/PMOVES.AI/blob/003e810b2211ebe2074d21bcfecbdf27ba92bedf/README.md

[2] [3] [11] [12] [13] [14] wizard.sh
https://github.com/POWERFULMOVES/PMOVES.AI/blob/003e810b2211ebe2074d21bcfecbdf27ba92bedf/pmoves/scripts/install/wizard.sh

[4] env.example.diff
https://github.com/POWERFULMOVES/PMOVES.AI/blob/003e810b2211ebe2074d21bcfecbdf27ba92bedf/pmoves/pmoves_provisioning_pr_pack/patches/env.example.diff

[5] [6] [47] [49] env.shared.example
https://github.com/POWERFULMOVES/PMOVES.AI/blob/003e810b2211ebe2074d21bcfecbdf27ba92bedf/pmoves/env.shared.example

[7] [8] [18] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [37] [38] [39] [54] [55] [56] Makefile
https://github.com/POWERFULMOVES/PMOVES.AI/blob/003e810b2211ebe2074d21bcfecbdf27ba92bedf/pmoves/Makefile

[9] [10] README_APPLY.txt
https://github.com/POWERFULMOVES/PMOVES.AI/blob/003e810b2211ebe2074d21bcfecbdf27ba92bedf/pmoves/pmoves_provisioning_pr_pack/README_APPLY.txt

[16] [17] mini_cli.py
https://github.com/POWERFULMOVES/PMOVES.AI/blob/003e810b2211ebe2074d21bcfecbdf27ba92bedf/pmoves/tools/mini_cli.py

[19] [42] README.md
https://github.com/POWERFULMOVES/PMOVES.AI/blob/003e810b2211ebe2074d21bcfecbdf27ba92bedf/pmoves/docs/services/wger/README.md

[36] [40] [41] [43] [44] [45] [46] [48] [57] [58] EXTERNAL_INTEGRATIONS_BRINGUP.md
https://github.com/POWERFULMOVES/PMOVES.AI/blob/003e810b2211ebe2074d21bcfecbdf27ba92bedf/pmoves/docs/EXTERNAL_INTEGRATIONS_BRINGUP.md

[50] [51] [52] [53] env_check.sh
https://github.com/POWERFULMOVES/PMOVES.AI/blob/003e810b2211ebe2074d21bcfecbdf27ba92bedf/pmoves/scripts/env_check.sh