



Centelligence Brain Bot: Free Setup Guide

Centelligence Brain Bot, a Human AI Brain you set up in your own Grok Bot.

Free: the bot and this guide cost nothing. To use them you need Grok Bot (a paid plan). The PLM runs on your own Modal account, which needs a card on file and can charge you once you pass its monthly included credit.

From NeuralSI.

This is the free guide for the free bot. It's separate from the paid Centelligence Brain guide, and you don't need that guide for anything here.

Contents

1. [Part 0: Before you start](#)
 1. [What you need](#)
 2. [How long it takes](#)
 3. [What this guide covers](#)
 4. [Rules that hold the whole way through](#)
 5. [Privacy, in plain words](#)
2. [Part 1: Add the bot](#)
3. [Part 2: Your setup conversation](#)
 1. [What it does, in order](#)
 2. [Tips](#)
 3. [Where your records live](#)
4. [Part 3: Setup prompts to paste](#)
 1. [P-1, "Start setup"](#)
 2. [P-2, "Import my people"](#)
 3. [P-3, "What do I owe here?"](#)
 4. [P-4, "Recall"](#)
 5. [P-5, "That's wrong"](#)
 6. [P-6, "Checkpoint"](#)

7. [P-7, "Pause, resume, recover, undo, restore"](#)
8. [P-8, "Turn on my routines"](#)
9. [P-9 to P-29: GitHub, drive and your PLM](#)
5. [Part 4: GitHub \(private repository\)](#)
 1. [Why](#)
 2. [4.1 Make your GitHub account](#)
 3. [4.2 Connect GitHub to your bot](#)
 4. [4.3 P-9, "Make my private GitHub repo"](#)
 5. [4.4 Before every push](#)
 6. [4.5 After that](#)
 7. [4.6 If you made it public by mistake](#)
 8. [4.7 How to stop](#)
6. [Part 5: Back up to your drive](#)
 1. [5.1 Connect your drive](#)
 2. [5.2 P-10, "Back up my brain folder"](#)
 3. [5.3 Schedule](#)
 4. [5.4 P-11, "Restore from my drive"](#)
7. [Part 6: Build your own PLM](#)
 1. [6.1 What a PLM is, and what it isn't](#)
 2. [6.2 What you need](#)
 3. [6.3 Set up Modal and connect it](#)
 4. [6.4 Get the code and connect Modal](#)
 5. [6.5 Make your tokens \(P-14\)](#)
 6. [6.6 Deploy \(P-15\)](#)
 7. [6.7 Load your notes and check its health](#)
 8. [6.8 The nightly job and the run caps](#)
 9. [6.9 Connect your bot to your PLM](#)
 10. [6.10 Pause or turn it off \(P-20\)](#)
 11. [6.11 Your contacts table](#)
 12. [6.12 What hasn't been tested yet](#)
8. [Part 7: Daily use](#)
 1. [A normal day](#)
 2. [Each week](#)
 3. [When something feels off](#)
9. [Part 8: Troubleshooting](#)
 1. [The bot and its files](#)

2. [GitHub, drive and PLM](#)
10. [Part 9: Plain facts and limits](#)
 1. [Cost](#)
 2. [Your data](#)
 3. [What the bot does and doesn't do](#)
 4. [No promises](#)
 5. [Names](#)
11. [Appendix: If the starter isn't available](#)
 1. [P-13, "Write my PLM starter"](#)

Part 0: Before you start

What you need

- A Grok Bot account on a paid plan. The bot runs inside it. Not every paid plan includes Grok Bot. Check cursor.com/help/grok-bot/plans for which ones do (as of Oct 2026).
- A GitHub account. The free plan is fine. (Part 4)
- A Google Drive or OneDrive account for backups. (Part 5)
- A Modal account with a card on file, for your PLM. (Part 6)

Grok Bot, GitHub, your drive and Modal are separate services. Each has its own terms and prices. Part 9 has the plain facts on cost.

How long it takes

We haven't timed a full setup yet.

What this guide covers

This is the complete setup. Bot plus guide gets you all of it:

1. The bot and its setup conversation (Parts 1 and 2)
2. The setup prompts you paste in (Part 3)
3. A private GitHub copy of your notes, contacts and code (Part 4)
4. Backups to the drive you choose (Part 5)
5. Your own PLM, built on your own Modal account (Part 6)

The PLM is part of the complete setup. You don't have to finish it before you start. Your bot works from day one, and it gets more out of your notes once the PLM is in place.

The bot is a helper. It doesn't replace you. You lead. It helps.

Rules that hold the whole way through

1. Keys and tokens never go into chat, notes, commits or backups. Your bot keeps them in private files on its own computer and never prints them.
2. Your GitHub repository stays PRIVATE. Always.
3. Your bot shows you each step and waits for your yes before it creates, uploads, deploys or pushes anything.
4. Back up before anything gets overwritten.

Privacy, in plain words

Your records live in plain files on the bot's computer. Other assistants you run on that same computer may be able to read them. The bot tells you this during setup.

Never give the bot passwords, card numbers, ID numbers or account numbers. It won't save them.

Part 1: Add the bot

1. Go to the Centelligence Brain Bot page at <https://neuralsi.tech/brain> and click "Add the bot". (What happens after you click hasn't been confirmed on a new account yet.)
 2. Open it. You should see it listed by name, ready for a message. (Not yet confirmed on a new account.)
 3. Rename it if you like. (We haven't checked whether renaming changes anything.)
-

Part 2: Your setup conversation

The bot runs setup the moment you start talking to it. You don't need a special command. If it doesn't start, use P-1 in Part 3.

What it does, in order

1. It introduces itself and explains how it works. It labels what it tells you as FACT, INFERENCE or UNKNOWN, tells you which file each fact came from, and asks for your yes before it saves or changes anything.
2. It checks its own computer. It looks for its home folder and checks for git and python3. It asks you nothing during this step.
3. If it finds a brain folder from before, it stops and asks you to pick: pick it up, start fresh in a new folder, or stop. It never overwrites the old one.
4. It asks nine questions, one per message:
 1. What to call you, and your time zone
 2. The three things that matter most to you right now
 3. The goal for each of those three, and a date if there is one
 4. The people involved: name, what they do, how you reach them
 5. Anything already decided
 6. What would make this week a win
 7. What tends to get dropped
 8. A few messages you've written, so it can draft in your voice (optional)
 9. Which routines to turn on, and at what times
5. It shows you the plan: the folder it will make, and every row and note it will add from your answers. It creates nothing until you say yes.
6. After your yes, it builds the folder, fills it in, saves a dated checkpoint, and reads back a short summary of every file.
7. It hands off with a few phrases you can use any time: "what do I owe here?", "recall <name or topic>", "that's wrong, it's this", "pause", "resume", "recover", "undo" and "restore".

Tips

- Short answers are fine. You can change anything later. Say "that's wrong, it's this."
- Nothing is lost if you stop halfway. (How the bot picks up a half-finished setup is not yet tested.)

Where your records live

The bot keeps a folder called the brain folder on its own computer. By default it's `~/centelligence-brain/`. Those files are the source of truth. If the bot's memory and a file disagree, the file wins.

File or folder	What's in it
now-card.md	What matters right now

File or folder	What's in it
tasks.csv	Projects and tasks
decisions.csv	Decisions, in your own words
loops.csv	Promises and asks you're tracking
people/contacts.csv	One row per person or organization
people/notes/	One note per person or organization
people/interactions.csv	What happened with whom, and when
notes/	Projects and topics
checkpoints/	Dated snapshots of where things stand
state.md	Line 1 says whether writing is on or paused
log/changes.log	One line for every change

If git is on the bot's computer, the folder is also a local git repository. Each change you approve gets its own version, so you can undo it. These versions sit on the same computer, so they don't protect you if that computer is lost. Parts 4 and 5 fix that.

Part 3: Setup prompts to paste

Paste these into your bot's chat. Each one is plain words. Your bot shows you what it plans to do and waits for your yes.

Prompts P-9 to P-29 belong to Parts 4 to 6, except P-13, which is in the Appendix. They're listed at the end of this part and written out in full where you use them.

P-1, "Start setup"

Use this if the bot didn't start setup on its own.

Please run your setup with me now. Check your computer, then ask me your setup questions one at a time.

What you should see: the bot's introduction, a short report on its computer, then question 1. (Not yet tested on a fresh setup.)

P-2, "Import my people"

Here's a list of people I work with (pasted below). For each one, propose a contact row and a short note. Ask me "same person as ...?" if anyone looks like someone already on file. Show me everything before you save.

What you should see: proposed rows and one note per person, with any likely duplicates flagged. Nothing saved until you say yes.

P-3, "What do I owe here?"

What do I owe here?
<paste the email, meeting notes, calendar entry or chat>

What you should see: a list of the explicit promises and asks in what you pasted. Each one shows who owes it, to whom, the exact quote, and a due date if one was stated. It offers to track them, and to draft a reply that stays in the chat.

P-4, "Recall"

Recall <name or topic>

What you should see: what's on file, each line labeled FACT, INFERENCE or UNKNOWN, with the file it came from and the date it was last confirmed. If nothing is on file, it says so and lists what it checked.

P-5, "That's wrong"

That's wrong, it's this: <the correct fact>

What you should see: every place the old fact appears, shown as old and new. It changes them on your yes and reads them back.

P-6, "Checkpoint"

Checkpoint

What you should see: a short summary of today (what changed, what got done, decisions, promises opened or closed, people updated, what's next) and suggested changes to your Now card. It saves a dated checkpoint on your yes.

P-7, "Pause, resume, recover, undo, restore"

Pause

Resume

Recover

Undo

Restore

What you should see:

- Pause: it turns writing off and reads that back. It can still answer you.
- Resume: it turns writing back on.
- Recover: a short picture of where things stand, plus one suggested next step. It changes nothing.
- Undo: it shows your last change and puts things back the way they were, on your yes.
- Restore: it lists recent versions and what would change, and restores on your yes.

P-8, "Turn on my routines"

Turn on my routines: <morning brief / evening checkpoint / nightly records check / Friday review>, at <times>.

What you should see: each routine you picked confirmed with its schedule in your time zone. The others stay off. (Turning routines on after setup with this prompt is not yet tested on a fresh setup.)

P-9 to P-29: GitHub, drive and your PLM

#	Name	Where
P-9	Make my private GitHub repo	Part 4
P-10	Back up my brain folder	Part 5
P-11	Restore from my drive	Part 5
P-12	Connect Modal	Part 6.4
P-13	Write my PLM starter	Appendix
P-14	Make my PLM tokens	Part 6.5
P-15	Test and deploy	Part 6.6
P-16	Load my notes and contacts	Part 6.7
P-17	PLM health check	Part 6.7
P-18	Use my PLM when you answer	Part 6.9
P-19	Keep my PLM current	Part 6.9
P-20	Pause my PLM	Part 6.10
P-21	Set up my contacts table	Part 6.11
P-22	Import my contacts	Part 6.11
P-23	Merge duplicates	Part 6.11
P-24	Who is <name>?	Part 6.11
P-25	Run the starter tests	Part 6.4
P-26	Set my app name and time zone	Part 6.4
P-27	Check it's up	Part 6.6
P-28	Show held items	Part 6.8
P-29	Put the starter in my brain folder	Part 6.4

The commands in prompts P-14 to P-17 and P-25 to P-29 were tested on Oct 24, 2026, in a fresh copy of the starter with made-up notes and contacts, deployed to an existing Modal account (not a brand-new

one). P-12 was tested up to the Modal sign-in link. Signing up for Modal, approving that link, and a new account's first bill weren't tested. (The other prompts are not yet tested on a fresh setup.)

Part 4: GitHub (private repository)

Why

Your bot's versions sit on its own computer. A private GitHub repository gives you a copy somewhere else, with full history.

You have one private repository, and it's your brain folder. It holds your notes, your contacts, your brain files, and your PLM's code in `plm/` inside the brain folder. Your PLM reads your notes and contacts from it (Part 6). That's why this comes before the PLM.

The repository's `.gitignore` keeps tokens, `.env` files, databases and the PLM index out.

"Private" is a GitHub setting, not a promise that no one else can ever see the files. GitHub handles your repository under its own terms, and anyone you add as a collaborator, or any app you give access, can read it. Keep two-factor sign-in on, and don't give other apps access you don't need.

A `.gitignore` only stops new files from being added. It doesn't remove anything already committed. Before your first push, and if you ever change the `.gitignore`, ask your bot to show you what's already tracked. Never put passwords, tokens, API keys or card numbers in any file in the brain folder. If one ever gets committed, change it right away, because it stays in the history.

Your contacts and notes are about other people. Keep only what you need and are comfortable holding, and don't add anything sensitive about someone (health, money, ID numbers) that they wouldn't expect you to keep.

Because it holds your notes and contacts, the repository must stay PRIVATE. Your bot checks that it's private before every push, and scans for passwords, tokens and keys before every push.

4.1 Make your GitHub account

1. Go to <https://github.com> and sign up. The free plan is fine.
2. Turn on two-factor sign-in in your GitHub settings. (GitHub moves this setting from time to time, so the menu wording may differ.)

4.2 Connect GitHub to your bot

Your bot signs in with GitHub's command-line tool, `gh`. It runs `gh auth login` and shows you a one-time code. You open <https://github.com/login/device> and type that code there. Your bot never sees your password.

If `gh` isn't on your bot's computer, your bot installs it. (Whether it's already there on a new bot isn't confirmed yet.)

Grok Bot also has a GitHub connector. Creating a private repository and pushing files with it is not yet tested, so use `gh` for now.

4.3 P-9, "Make my private GitHub repo"

Set up a PRIVATE GitHub repository for my brain folder. Step by step, and wait for my yes before each step:

1. Check that the brain folder is a git repository with at least one commit, and that git's name and email are set for this repository, not globally.
 2. Add a .gitignore that keeps out tokens, keys, .env files, databases (*.db, *.sqlite), the PLM index, Python environments (.venv), the PLM's own log files (plm/**/*.*.log), and the snapshots folder. Keep log/changes.log in git: it's the brain's record of what changed.
 3. Scan every file you would commit for anything that looks like a password, token, key, or account number. Show me the file names it flags (never the values). Stop if anything is flagged.
 4. Help me sign in to GitHub with ``gh auth login`` (device code). I'll type the code myself.
 5. Create a PRIVATE repository named <pick-a-name> with ``gh repo create <pick-a-name> --private --source=<brain folder> --remote=origin --push``.
 6. Show me proof it's private (``gh repo view --json visibility``) and the latest commit, then I'll check it on github.com myself.
- Never make it public. Never force-push. Never commit secrets.

What you should see: the scan result, the device code, then "visibility: PRIVATE" and a commit id that matches what you see on github.com. If it didn't work: if the scan flags a file, stop and deal with that file before anything is pushed. (Other error messages haven't been tested yet.)

4.4 Before every push

Your bot does two checks each time, before it pushes:

1. Privacy check: it confirms the repository is still private (`gh repo view --json visibility` says PRIVATE). If it isn't, it stops and tells you.

2. Secret scan: it scans the new commits for passwords, tokens, keys and account numbers. If anything is flagged, it stops.

Pushes always go forward. Your bot never force-pushes or rewrites history. If a push is refused, it stops and explains.

(Not yet tested on a fresh setup.)

4.5 After that

By default, your bot asks before each push. Once you're comfortable, you can say "push after each checkpoint."

(The exact wording the bot accepts for this is not yet tested on a fresh setup.)

4.6 If you made it public by mistake

1. On github.com, open the repository, then Settings, then Change visibility, and set it back to Private. (GitHub may word this menu path differently.)
2. Treat anything that was in it as seen. Change any password or token that was ever in those files.

4.7 How to stop

- Say "stop pushing to GitHub." Your bot removes the remote from the brain folder, on your yes.
- Sign out on the bot's computer: `gh auth logout`.
- On github.com you can archive or delete the repository yourself. Your bot never deletes it.

If you ask your bot to forget something that was already pushed, it stays in GitHub history until you remove it there.

Part 5: Back up to your drive

Grok Bot can copy a file from your bot's computer into your connected Google Drive or OneDrive. Each backup is a new dated file. Older backups are never overwritten or deleted.

5.1 Connect your drive

1. In Grok Bot, connect Google Drive or OneDrive. (The exact steps for a new account, and whether a work account behaves differently from a personal one, aren't confirmed yet.)
2. Dropbox or another drive: there's no tested upload path yet. Until there is one, download the backup file yourself and drop it into that drive.

5.2 P-10, "Back up my brain folder"

Back up my brain folder to my <Google Drive or OneDrive>, in a folder called "Brain Backups". Show me the plan and wait for my yes:

1. Make one dated archive of the brain folder (name it brain-backup-YYYY-MM-DD-HHMM.tar.gz) plus a checksum file. Leave out tokens, keys, .env files, databases (*.db, *.sqlite), the PLM index, Python environments (.venv), and anything outside the brain folder.
2. Upload both files to "Brain Backups". Never overwrite or delete an older backup; each backup is a new dated file.
3. Read back the file names and sizes that landed, and tell me where to look.

What you should see: two new dated files in "Brain Backups", with sizes that match what the bot reports. Open the folder yourself and check. (Drive error messages haven't been tested yet.)

5.3 Schedule

Backups are off until you turn them on. A simple habit: one backup each Friday, as part of your Friday review, after your yes.

(The exact wording the bot accepts for a Friday backup is not yet tested on a fresh setup.)

5.4 P-11, "Restore from my drive"

Restore my brain folder from my drive backup <file name>. Download it, check the checksum, unpack it into a NEW folder next to the current one, show me what's different, and switch over after my yes, not before. Keep the current folder as it is.

What you should see: the checksum result, the new folder, a list of what's different, then a question before it switches over. (Restore is not yet tested end to end on a fresh setup.)

Your PLM's search index doesn't need its own backup. It can be rebuilt from your notes (see 6.7).

The backup lands in your own drive account, under that provider's terms.

Part 6: Build your own PLM

6.1 What a PLM is, and what it isn't

PLM stands for Personal Language Model. In this guide it means your own small search service in the cloud. It reads your notes, turns them into searchable data on its own server, and answers your bot's questions with short excerpts from your notes. Each excerpt names the file it came from.

It isn't the bot's memory, and it isn't the brain folder from Part 2. Those stay where they are. The PLM is a separate service you own and run on your own Modal account. You can point it at your brain folder, at a notes folder of your own, or both.

Your bot works before your PLM is done. The PLM is part of the complete setup, so plan to finish it.

The parts, all in your own accounts:

Part	What it does	Where it runs
Notes repository	Your notes as plain files (.md, .txt) in a git repository. The source of truth.	Your bot's computer, copied to your private GitHub repo
Contacts table	One row per person with a stable contact ID, joined to that person's note, so contacts are searched together with their notes (6.11).	Your notes repository, indexed in your Modal app
Modal app	Your cloud service. Starts when called and shuts down when idle (scale to zero).	Your Modal account
Embeddings	<code>nomic-embed-text-v1.5</code> , run locally on CPU inside your Modal container with fastembed (ONNX). The model runs inside your own app, so there's no per-word bill.	Inside your Modal app
Vector store	One SQLite file holding text chunks, their embeddings, and a keyword index (FTS5). Search mixes meaning and keywords.	A Modal Volume (a cloud disk)
API with a token	<code>GET /v1/health</code> (no token; counts, run times and your time zone, no note text), <code>POST /v1/ask</code> (search), <code>POST /v1/ingest</code> (send changed notes). Each caller has its own token; the server keeps a SHA-256 fingerprint of each token, never the token.	Your Modal app

Part	What it does	Where it runs
Ingest	After your bot saves and commits notes, a small script sends the changed files to <code>/v1/ingest</code> .	Your bot's computer
Scheduled job	One Modal cron job each night: catches up anything that was held back, and applies deletions from your weekly list.	Your Modal app
Run caps	Limits that help keep costs down (6.8)	Built into the code

6.2 What you need

- A Modal account with a card on file. Modal requires a payment method. Part 9 has the cost facts.
- A budget, set before anything else (step 3 below).
- Your private GitHub repository from Part 4.

6.3 Set up Modal and connect it

1. Sign up at <https://modal.com> and pick the Starter plan.
2. Add a payment method (Settings, Usage & Billing). Modal requires one.
3. Set a budget before anything else. Go to Settings, Usage & Billing, then set a Workspace budget (a monthly usage cap) and, if you like, a spend limit (a cap on what you pay after included credits). When the spend limit is reached, Modal stops work that would cost more. Pick a number you're comfortable with. That's your call.
4. You connect your bot's computer to Modal in 6.4 (P-12), once the starter is in place. Modal shows a sign-in link. You open it and approve. Your bot never sees your Modal password.

6.4 Get the code and connect Modal

Your PLM code comes from a free starter. You copy it into `plm/` inside your brain folder, so you have your own copy, under the starter's license, and it lives in your one private repository from Part 4, under the same rules. There's no second repository. In the prompts below, "the starter folder" means `plm/`.

1. Get the starter at <https://github.com/leviweck/centelligence-starter-plm> and copy it with git clone, or with GitHub's Download ZIP button.
2. Paste P-29. Your bot puts the starter in `plm/` as plain files, commits it, and shows you that your repository is still private.
3. Paste P-12 to connect Modal.

The `.gitignore` keeps tokens, `.env` files, databases and the PLM index out of the repository.

If the starter isn't available, see the Appendix.

P-29, "Put the starter in my brain folder"

Copy the PLM starter into a folder named plm inside my brain folder (<brain home>/plm/). If you use git clone, clone it straight into <brain home>/plm/, then remove the plm/.git folder (it only holds the starter's own history) so plm/ is saved as normal files in my brain repository, not as a nested repository. If you use the ZIP, unzip it and rename the unzipped folder to plm. Don't create a separate repository and don't clone anything outside my brain folder. Commit it to my brain repository and show me that ``git ls-files plm`` lists the starter's files. Then show me ``gh repo view --json visibility`` for my brain repository so I can see it says PRIVATE.

What you should see: plm/ committed in your brain folder as normal files (`git ls-files plm` lists them), then `gh repo view --json visibility` showing PRIVATE. The `gh repo view` check works once you've made your private repository with P-9 in Part 4. (This check is not yet tested on a fresh setup.)

P-12, "Connect Modal"

Your bot makes a Python environment for your PLM in plm/.venv. The starter's own .gitignore keeps it out of git, so it stays out of GitHub.

Set up Modal for my PLM inside <brain home>/plm/. Make a Python environment there named .venv (the starter's .gitignore keeps it out of git; check ``git status`` shows no .venv files). Run ``pip install -r requirements.txt``, then run ``modal token new`` and give me the link so I can approve it in my browser. Then run ``modal profile list`` and show me the workspace name. Never print or save the Modal token anywhere else.

What you should see: a sign-in link to approve in your browser, then your Modal workspace name. `git status` shows no .venv files. If no browser opens on the bot's computer, Modal prints a link for you to open yourself. (Approving that link and finishing the sign-in is not yet tested on a fresh setup.)

P-25, "Run the starter tests"

These tests run on your bot's computer. They don't touch Modal.

In <brain home>/plm/, run ``pip install -r requirements-dev.txt`` and then ``pytest -q``. Tell me how many passed. Then run ``pip install fastembed`` and ``python scripts/local_check.py``, and show me the questions and the top file for each. Don't deploy anything.

What you should see:

- "16 passed".
- Four sample questions, each answered from the right sample note.
- Two merged contacts, each landing on the right person.
- The checks listing one contact with no note, one note with no contact, and one likely duplicate.

The local check downloads the embedding model, about half a gigabyte. It may download again later, because it's kept in a temporary folder. The model file comes from Hugging Face; your notes aren't sent there. (How long the download takes on your bot's computer isn't measured yet.)

P-26, "Set my app name and time zone"

Set `NS_APP_NAME` to <short name> and `NS_TIMEZONE` to my time zone (for example `America/Chicago`). Save these two settings in my notes for next time. They are not secrets.

Your time zone decides when "today" starts for the daily run cap, and when the nightly job runs (00:45 your time). Your bot needs `NS_APP_NAME` and `NS_TIMEZONE` set in the same session when it runs `admin.py` (P-14) and the deploy (P-15), and `--app` must match your app name. If they aren't, the deploy falls back to the default name and UTC time.

6.5 Make your tokens (P-14)

Your PLM uses two tokens: a read token for searching and a write token for sending notes. Your bot makes them, keeps them in private files outside your brain folder, and never shows them to you or prints them. So they stay out of GitHub and out of your backups.

In <brain home>/plm/, run ``python scripts/admin.py make-tokens --app <app name>``. It saves a read token and a write token under `~/.config/<app name>/` (owner-only files) and uploads only their fingerprints to Modal. Never show me or print the tokens. Then show me ``modal secret list``.

What you should see: the output of `modal secret list`, showing your secret by name. No token values anywhere. Owner-only files keep other user accounts on that computer out. Other assistants or programs you run on the same computer may still be able to read them (see Part 0). Before this runs, your bot sets `NS_APP_NAME` and `NS_TIMEZONE` in the same session, and `--app` is your app name (see P-26).

6.6 Deploy (P-15)

Your bot runs the tests and the deploy from `plm/` inside your brain folder.

Run the tests again. If they pass, show me what ``modal deploy app.py`` will create: one app, two volumes (`<app name>-index` and `<app name>-state`), the secret, one nightly job, and a web address. Wait for my yes, deploy, and give me the web address that ends in `modal.run`. Save the address and the token file paths as pointers in your memory, never the tokens.

What you should see: the list of what it will create, a question before it deploys, then a web address that ends in `modal.run`. Keep your web address to yourself. Anyone who has it can open the health check, and each visit can wake your search service and use a little Modal compute. Before the deploy, your bot sets `NS_APP_NAME` and `NS_TIMEZONE` in the same session (see P-26). If it doesn't, the deploy falls back to the default name and UTC time. Your opening deploy downloads the embedding model into the image, so it takes a while. Your web address looks like `https://<workspace>--<app name>-api.<region>.modal.run`. (How long an opening deploy takes on a new account isn't measured yet.) If it didn't work: if the tests fail, nothing gets deployed. Ask your bot to explain the failure. (Common deploy errors haven't been seen in testing yet.)

P-27, "Check it's up"

Open `<web address>/v1/health` and tell me in plain words if it's up and how many notes it has.

What you should see right after the deploy: up, 0 notes. The opening check can take several seconds while your app starts.

6.7 Load your notes and check its health

P-16, "Load my notes and contacts"

`push.py` lives in `plm/` inside your brain folder.

From `<brain home>/plm/`, do a dry run first: ``python scripts/push.py --source <brain home> --url <web address> --token-file ~/.config/<app name>/write.token --dry-run``. Show me the file count and size. Wait for my yes, run it without `--dry-run`, then check `/v1/ingest/status` until pending is 0.

What you should see: a file count and size from the dry run, then a status that reaches pending 0.

Good to know:

- The source can be a git repository or a plain folder. From a git repository, `push.py` sends what changed since the last send, read from the last commit. Edits you haven't saved don't go.

- push.py skips plm/ for you, so the starter's own files never get sent as notes.
- `--include` is optional. Use it only to send just some folders, for example `--include notes,people`. When you use it, leave plm out.
- The contacts file always goes when it changes.
- Your contacts file is people/contacts.csv unless you say otherwise with `--contacts <path>`.
- To rebuild the index from scratch later, run a full send again.

P-17, "PLM health check"

Check my PLM: call `/v1/health` and `/v1/ingest/status` (write token) and tell me in plain words: is it up, how many notes, chunks and contacts, when it last updated, how many runs it used today out of 24, anything held for my yes, and any contact checks. Then ask it one test question about something you know is in my notes and show me the top excerpt and its file.

Healthy looks like:

- It's up.
- The note count is close to your file count, and the contact count matches your contacts file.
- It updated within the last day.
- The test question brings back the right file.

6.8 The nightly job and the run caps

These rules help limit how often your PLM runs, and so what it costs.

- One Modal cron job runs at 00:45 in your time zone. Modal's Starter plan allows 5 deployed cron jobs, and this uses one (as of Oct 2026; check modal.com/pricing).
- Up to 24 update runs a day, counted midnight to midnight in your time zone.
- At least 10 minutes between runs. A change that arrives in between waits: push.py waits out the gap and then asks for a run (unless you add `--no-wait`). If push.py is stopped before then, the change runs with your next send after the gap, or at 00:45 in your time zone, whichever comes first. (push.py asking for the run on its own is not yet tested on a fresh setup.)
- Up to 120 sends an hour per token. Past that, your PLM answers "rate limited" and push.py retries.
- Over the daily cap, nothing is rejected. The change waits for the 00:45 run, and no extra run is started for it. That's why a change sometimes doesn't show up until morning.
- Your deploy sets up the nightly job, and it does its work when started by hand. (The 00:45 run starting on its own hasn't been seen in testing yet.)

These caps are rules in the starter's code, not Modal limits. You can change them in the starter's settings. Your costs move with them.

Held items

Some changes wait for your yes:

- more than 50 deletes in one send;
- more than 5% of notes missing from the weekly list;
- a big drop in contacts (more than 20% in one save, once you have 10 or more contacts).

A weekly list older than 26 hours deletes nothing.

If you just sent with `--manifest`, a note you deleted can still be listed as held even though it's already gone from search. `--release deletes` clears the list at the next run, but it also applies every other delete on that list, so check the list first (P-28). The old reason can still show after the list is empty.

P-28, "Show held items"

Show me what `/v1/ingest/status` says is held and why. If I say yes, run `push.py` with `--release deletes` (or `--release contacts`).

What you should see: each held item and why it's held, then a question before anything is released.

6.9 Connect your bot to your PLM

P-18, "Use my PLM when you answer"

From now on, when my brain files don't answer a question about my own notes, search my PLM with the read token (`POST /v1/ask`, excerpts) and label what comes back FACT with the file path it came from. My brain files still win when they disagree. If the PLM is down, say so and answer from the files.

What you should see: the bot confirms the rule. Later answers that use your PLM show FACT with the file path.

P-19, "Keep my PLM current"

After each commit to my brain folder, run `push.py` in the background so my PLM gets the change. Don't wait for it. Each night during the records check, tell me if anything failed to send.

What you should see: the bot confirms it. From then on, saved changes reach your PLM within minutes, or by 00:45 if the day's cap is used.

Keeping it current uses the P-16 command without --dry-run after each commit. Once a week, add `manifest`, so notes you deleted leave the index.

(Whether the bot accepts P-18 and P-19 is not yet tested on a fresh setup.)

Your brain files still win when they disagree with your PLM.

6.10 Pause or turn it off (P-20)

`modal app stop <app name>` stops the app and its nightly job.

Pause my PLM: stop sending changes, and show me the command to stop the Modal app (``modal app stop <app name>``). Wait for my yes before you run it.

What you should see: sending stops, then the stop command, then a question before it runs. Stopping the app doesn't delete your volumes or secret. They stay in your Modal account until you delete them.

6.11 Your contacts table

Your PLM keeps one table of the people and organizations you deal with. Each one gets a contact ID that never changes, and each contact ID points to that person's note. When your PLM indexes your notes, it joins the two. So a search about a person brings back who they are and what your notes say about them.

You already have this table if you finished Part 2: `people/contacts.csv`, with one note per person in `people/notes/`. Your PLM reads that same file. Nothing gets copied by hand.

(The bot-side prompts P-21 to P-23 and the Google Sheet route are not yet tested on a fresh setup.)

The table

File: `people/contacts.csv` in your notes repository. Plain CSV, UTF-8, with the header in the top row. A Google Sheet works too. Your bot reads the sheet and saves it as this CSV, and the CSV is what the PLM reads.

Column	Required	What goes in it
contact_id	yes	Stable ID. People: c-firstname-lastname. Organizations: o-name. Add -2, -3 for a second person with the same name. Never changed, never reused.
name	yes	Full name or organization name
organization	no	Where they work
role	no	Their job or role

Column	Required	What goes in it
type	no	person or organization
reach_by	no	How you usually reach them (email, phone, text, in person)
email	no	Kept in the table; never embedded and never put in search excerpts
phone	no	Stored as text; never embedded and never put in search excerpts
stage	no	met / talked / working together / opportunity / outcome
identity	no	confirmed / owner-said / unconfirmed / UNKNOWN
last_confirmed	no	YYYY-MM-DD, the last time a fact about them was confirmed
source	no	Where the facts came from
note_path	no	people/notes/<contact_id>.md
updated_at	no	YYYY-MM-DD
merged_into	no	Blank, or the contact_id of the row this one was merged into

Your PLM needs the contact_id column. A file without it is refused with a plain message. The other columns are optional for your PLM, though your bot fills in name and note_path for everyone.

Each person's note starts with a short header that names its contact ID:

```
---
contact_id: c-jane-doe
---
```

(The merged_into column in the bot's own contacts layout is not yet tested on a fresh setup.)

How your PLM joins contacts to notes

1. After a commit, push.py sends people/contacts.csv as a whole file whenever it changed, along with any changed notes.
2. The server refuses a contacts file without a contact_id column, and says why.
3. Each row is stored in a contacts table keyed on contact_id. A row that didn't change isn't embedded again.

4. Each active row (merged_into blank) becomes one searchable contact entry, built from name, organization, role, type, stage and reach_by. Email and phone are left out.
5. Each note whose header names a contact_id has its excerpts tagged with that ID.
6. The join runs both ways. A search hit on a note shows the person's contact card. A search hit on a contact shows the path to their note.
7. Problems get reported, never fixed silently. The status check and health check show:
 - orphans: a row whose note_path doesn't exist, or a note whose contact_id has no row;
 - likely duplicates: same email, same phone, or same name plus organization, on rows that aren't merged;
 - broken redirects: merged_into pointing to an ID that doesn't exist, or a loop (a to b to a).
8. A row that disappears from the file drops out of search, but nothing is deleted from your repository. Git history keeps it. If you have 10 or more contacts and one save drops more than 20% of them, the change is held for your yes, like large note deletes (6.8).

How a contact shows up when you ask

Ask "who is Jane Doe?" or use her contact ID. Your bot calls `<web address>/v1/contact/<name or ID>` with the read token and shows something like:

```
c-jane-doe: Jane Doe, Operations Lead at Example Co (person)
Stage: talked. Identity: confirmed. Last confirmed: 2026-10-01
Reach by: email
Note: people/notes/c-jane-doe.md
Merged here: c-jane-d
From her note: <top 2 or 3 excerpts, each with its file path>
```

A name has to match the name column exactly, though capital letters don't matter. If two people share a name, a name lookup brings back just one of them. Look people up by contact ID to get the right one. Each line is labeled FACT, with the file and contact ID it came from. If you look up an old ID or old name that was merged, you get the surviving card with a "merged from" line. Your brain files still win when they disagree with your PLM.

Duplicates and merges: nothing gets deleted

- Your PLM never merges anyone. It reports likely duplicates, and you and your bot decide.
- Merging happens on your yes. Pick the one to keep. Move the facts and sources to that row and note. Set merged_into on the old row to the kept contact_id (a cell that reads "merged into c-..." also works). Leave the old row and old note in place, with the old note cut down to one line: "Merged into c-...".
- The old ID keeps working. Lookups and searches for it land on the kept record. Chains (a merged into b, b merged into c) resolve to c.
- Nothing is deleted: not the row, not the note, not the history.

P-21, "Set up my contacts table"

Set up my contacts table for my PLM. Back up people/contacts.csv first. Check it has the header in section 6.11 (add an empty merged_into column at the end if it's missing), that every row has a contact_id and a note_path, and that every note in people/notes/ starts with its contact_id header. Show me every gap and the exact fix, and change nothing until I say yes.

What you should see: a list of every gap and its fix. Nothing changes until you say yes.

P-22, "Import my contacts"

Here's a list of my contacts (pasted below, or a CSV export from my phone, email or a spreadsheet). Map each column to the contacts table, give each new person a contact_id, and show me the new rows and notes before saving. Flag anyone who looks like someone already on file (same email, same phone, or same name and organization) instead of adding them twice. Never save passwords, card, ID or account numbers.

What you should see: new rows and notes to review, with possible repeats flagged instead of added twice.

P-23, "Merge duplicates"

Find likely duplicate contacts. For each pair, show me both rows, which one you'd keep, and exactly what would move. On my yes: move the facts to the survivor, set merged_into on the old row, and leave the old row and note in place as a pointer. Delete nothing.

What you should see: each likely pair side by side, with what would move. Nothing moves until you say yes.

P-24, "Who is <name>?"

Who is <name or contact_id>? Check my brain files first, then my PLM's contact lookup. Show the contact card, the note it links to, and the top excerpts, each labeled with its source.

What you should see: the contact card, the linked note, and the top excerpts, each with its source.

If your contacts live in a Google Sheet

My contacts live in the Google Sheet <name>. Read it with the Sheets connector and save it as people/contacts.csv, mapping its columns to the contacts table. Show me the result before saving.

(The Google Sheet route, its column mapping, and how often to refresh it are not yet tested.)

6.12 What hasn't been tested yet

The PLM steps in 6.4 to 6.11 were tested on Oct 24, 2026, in a fresh copy of the starter with made-up notes and contacts, deployed to an existing Modal account (not a brand-new one). These parts weren't:

- signing up for Modal, approving the Modal sign-in link, and a new account's first bill (6.3, P-12);
- how long the model download and the first deploy take on a new account (the test reused copies already on hand) (6.4, 6.6);
- the 00:45 nightly run starting on its own (6.8);
- push.py asking for a run on its own after the 10-minute gap (6.8);
- the daily cap, the hourly send limit, and the hold on more than 50 deletes (6.8);
- the hold on a big drop in contacts, and releasing held contacts (6.8, 6.11);
- the `gh repo view` check at the end of P-29;
- making new tokens to replace old ones (Part 8);
- the bot-side prompts P-18, P-19 and P-21 to P-23, and the Google Sheet route (6.9, 6.11);
- the Appendix.

Part 7: Daily use

A normal day

- Morning brief, if you turned it on.
- Through the day, talk to your bot as usual. Paste in emails, notes and chats and ask "what do I owe here?"
- Each change you approve gets saved and versioned. With P-19 on, your PLM gets it within minutes, or by 00:45 if the day's cap is used.
- Evening: say "checkpoint," or let the evening routine run. If you set it up, your bot pushes to GitHub after the checkpoint.

Each week

- Friday review, if you turned it on. A good time to run the drive backup (P-10) and the PLM health check (P-17).
- The nightly records check runs quietly, if you turned it on.

(How the nightly records check reports GitHub, drive backup and PLM status is not yet tested on a fresh setup.)

When something feels off

Say "recover." Your bot reads its files and gives you a short picture of where things stand, plus one suggested next step. It changes nothing while it does this.

Part 8: Troubleshooting

The bot and its files

What you see	What it means	What to do
The bot didn't start setup	It didn't detect a new setup	Paste P-1
"I found an existing brain folder"	A brain folder is already there	Pick it up, start fresh in a new folder, or stop. It won't overwrite the old one
It won't save anything	Writing may be paused (line 1 of state.md)	Say "resume"
It seems to have lost track	Time away, or a lost thread	Say "recover"
Something got changed by mistake		Say "undo," or "restore" to pick an earlier version

GitHub, drive and PLM

What you see	What it means	What to do
Health says up but 0 chunks	Nothing loaded yet	Run P-16

What you see	What it means	What to do
"queued_for_scheduled_run"	Day's 24 runs used	Nothing. It lands at 00:45
"waiting_for_gap"	A run happened less than 10 minutes ago	Nothing. push.py asks for the run when the 10 minutes are up. If push.py was stopped, your next send after the gap or the 00:45 run picks it up
"rate_limited"	120 requests in an hour	Nothing. push.py retries
401	Wrong or missing token file	Re-run P-14. It makes new tokens. The old ones stop working once your app restarts or is redeployed. If a token may have leaked, redeploy right away (P-15) so the old one stops working.
A name lookup shows the wrong person	Two people share that name	Look them up by contact ID
Deletes "held"	Big delete batch	Check what's being deleted, then say yes or no
Modal says budget reached	Your spend cap	Raise the budget, or wait for next month
GitHub push refused	Sign-in expired, or history differs	Ask the bot to explain. Never force-push
<code>git ls-files plm</code> shows one entry, not the starter's files	plm/ was added as a nested repository	Ask your bot to run <code>git rm --cached -f plm</code> , move plm/.git out of your brain folder, then <code>git add plm</code> and commit
Secret scan flags a file in plm/	A token, .env file, database or index file isn't covered by the .gitignore	Nothing gets pushed. Ask the bot to add it to the .gitignore and re-run the scan
The bot can't reach your PLM		Not tested yet. Ask your bot to explain what it sees

What you see	What it means	What to do
Answers don't name a note		Not tested yet. Ask your bot to explain what it sees

(Other GitHub, drive and Modal error messages haven't been tested yet.)

Part 9: Plain facts and limits

Cost

- The bot and this guide are free. There's no upsell inside the bot.
- Grok Bot (paid plan), Modal, GitHub and your drive are separate services with their own terms and prices.
- Not every paid plan includes Grok Bot. Check cursor.com/help/grok-bot/plans for which ones do (as of Oct 2026).
- Your bot's routines and setup work use your Grok Bot plan's usage. Past your plan's included usage, extra usage can cost money under Grok Bot's terms.
- Modal's Starter plan included \$30 of compute a month as of Oct 2026 (modal.com/pricing). Check their page for current prices.
- Other Modal list prices as of Oct 2026 (modal.com/pricing): CPU \$0.0000131 per physical core per second; memory \$0.00000222 per GiB per second; Volumes \$0.09 per GiB per month, with 1 TiB a month included. Check their page for current prices.
- Modal bills while containers load, while they work, and while they wait to shut down (60 seconds by default, and you can change it), as of Oct 2026 (same page). Check their page for current terms.
- The starter keeps your search service running for 5 idle minutes after each use, and Modal bills that time. Modal charges for whichever is higher, the CPU and memory your app asks for or what it actually uses (modal.com/pricing, as of Oct 2026). Check their page for current terms.
- In the starter, each update run and the nightly job ask Modal for 2 CPU cores and 2 GiB of memory, billed while they load and work plus their idle time before shutting down. The starter stops an update run after 30 minutes and the nightly job after 60. The search service asks for 1 CPU core and 2 GiB. Check the starter's `app.py` for the current sizes.
- What your PLM costs depends on how much you use it. (A typical monthly cost, and how big the index gets for your notes, aren't measured yet.)
- You can check your spend in Modal under Settings, Usage & Billing. (Whether the `modal billing` command's reports work on the Starter plan isn't confirmed.)

Your data

- Your notes and contacts are kept in accounts you control: your bot's computer, your private GitHub repository, your drive and your Modal account. This setup doesn't send them to NeuralSI, and NeuralSI has no access to those accounts. Grok Bot, GitHub, Modal and your drive provider handle your data under their own terms.
- The PLM starter's code sends your notes and contacts only to your own Modal app, where Modal stores and processes them as your provider. Your bot also pushes them to your own private GitHub repository and backs them up to your drive, each after your yes. Your own Grok Bot reads them to do its work. Apart from Grok Bot, this setup doesn't send them to any AI service: the search model runs inside your Modal app. Your search service is set up to load the model from your app's image, so it shouldn't need to download anything while it runs. Its packages and the model download when Modal builds your app's image, and the model also downloads to your bot's computer when you run the local check in 6.4.
- Other assistants you run on the bot's computer may be able to read your files.

What the bot does and doesn't do

- It never sends messages, posts, emails, buys, books or schedules anything for anyone. Drafts stay in the chat. You send. The outside actions it does take are the ones in this guide: pushing to your private GitHub repository, backing up to your drive, and deploying and updating your PLM on your Modal account, each after your yes. Running your PLM uses Modal compute, which can cost money past the included credit. (Not yet tested on a fresh setup.)
- It never saves passwords, card numbers, ID numbers or account numbers.

No promises

- No promised results. It's a helper, and you stay in charge.
- This guide and the bot are provided free and as is, without any warranty, to the extent the law allows. You set up and run everything in your own accounts, and you're responsible for what you approve, what your accounts cost, and the data you put in them. Grok Bot, GitHub, Modal and your drive provider can change their features, prices and terms at any time, and that can break a step. Check anything important before you rely on it. Nothing here is legal, financial, tax or other professional advice.

Names

- Grok, Grok Bot, X, GitHub, Modal, Google Drive, Google Sheets, OneDrive and Dropbox are trademarks of their owners, as are the other products named in this guide. They're named to say which services the steps work with. Not affiliated with or endorsed by SpaceXSI (formerly xAI), Anyosphere, Inc., X, GitHub, Modal Labs, Inc., Google, Microsoft or Dropbox.
-

Appendix: If the starter isn't available

Use this route if you can't get the starter in 6.4. Your bot writes the starter code from the spec in P-13, into plm/ inside your brain folder. It's the same design, but it takes longer and needs more checking.

(Not yet tested. A generated starter needs a code review before this guide can say it works.)

P-13, "Write my PLM starter"

Write a small PLM starter in the plm/ folder inside my brain folder (keep tokens, .env files, databases and the PLM index out of it; the .gitignore covers them). Show me each file before saving it. Spec:

- Python 3.12, Modal, FastAPI. One app. Pick a short app name with me.
- Image: debian_slim, pip install fastembed, and download nomic-embed-text-v1.5 into the image at build time. No GPU.
- core.py: split notes on headings, then into 800-character chunks with 150 overlap. Embed with nomic-embed-text-v1.5 (fastembed) using "search_document: " for notes and "search_query: " for questions. Store chunks, embeddings and an FTS5 keyword index in one SQLite file. Search = embeddings + keywords combined with reciprocal rank fusion; return the top 6 excerpts with file path and heading.

- app.py: one Modal Volume for the index, one for state. One Modal Secret holding a JSON map of token name -> SHA-256 of the token (never the token).

Routes: GET /v1/health (no token; counts and last-run times, never note text). POST /v1/ask (read token; question up to 2000 characters; returns excerpts). POST /v1/ingest (write token; files as base64 plus their git blob id; the server re-checks each blob id; deletes listed separately). GET /v1/ingest/status (write token).

A Writer (2 CPU, 2 GiB, max one container) applies pending ingest files. Caps: at most 24 Writer runs a day in my time zone, at least 10 minutes apart (requests in between get combined), at most 120 ingest requests an hour per token, 1 MB per request, 200 files per request. Over a cap: hold the request for the nightly run, never drop it. More than 50 deletes in one request, or more than 5% of notes missing from the nightly list: hold the deletes and ask me.

One cron job at 00:45 my time: apply held requests and deletes. API container: 1 CPU, 2 GiB, shuts down after 5 idle minutes.

- push.py (runs on your computer): after a commit, send the files that changed since the last sent commit (.md and .txt under the folders I name), read from git, not from the working folder. Queue failures and retry. Has --dry-run.

- admin.py: make two random tokens (read and write), save each in its own file with permission 600 under ~/.config/<app name>/, upload just their SHA-256 values to the Modal Secret with `modal secret create <name> --from-json <temp file> --force`, delete the temp file. Never print a token.

- Contacts (spec in section 6.11 of the guide): push.py also sends people/contacts.csv whenever it changes. The server checks the header has contact_id, upserts rows into a contacts table keyed on contact_id, embeds one contact chunk per active row (name, organization, role, type, stage, reach_by; never email or phone), and tags note chunks with the contact_id from the note's frontmatter. Rows with merged_into set are redirects: not embedded, but their

ID and name resolve to the survivor. Add GET /v1/contact/<contact_id> (read token): the contact card plus top note excerpts, following redirects. /v1/ask attaches the contact card (without email or phone) to any excerpt that carries a contact_id. /v1/ingest/status reports contact counts, likely duplicates and orphans. Never delete a contact row; never merge on your own.

- tests/smoke.py: fake notes and fake embeddings, no network. Must pass before deploy. Include fake contacts: one merged pair, one orphan note, one duplicate email.
- .gitignore that blocks tokens, keys, .env, *.db, *.sqlite, the PLM index, logs.

What you should see: each file shown to you before it's saved, then a smoke test you can run. After that, carry on from 6.5. (Not yet tested.)

Grok, Grok Bot, X, GitHub, Modal, Google Drive, Google Sheets, OneDrive and Dropbox are trademarks of their owners, as are the other products named in this guide. They're named to say which services the steps work with. Not affiliated with or endorsed by SpaceXSI (formerly xAI), Anysphere, Inc., X, GitHub, Modal Labs, Inc., Google, Microsoft or Dropbox.

© 2026 NEURALS

Grok, Grok Bot, X and GitHub are trademarks of their owners. Not affiliated with or endorsed by SpaceXSI (formerly xAI), Anysphere, Inc., X or GitHub.