

Dibab

User manual

Sort the administrative papers of a household, on your own machines

Version 0.9.0-alpha

2026-11-26

dibab.davalan.fr

Contents

Introduction **3**

What Dibab does . . . 3
What it does not do . . . 3
A tool for a household . . . 3
How this manual is organised . . . 4
Conventions . . . 4

Installation **5**

What you need . . . 5
The models, by role . . . 5
The final tree . . . 5
Installation from the Debian package . . . 6
Installation on Fedora, openSUSE Leap and Arch . . . 7
Installing from the sources . . . 7
Running the tests (optional) . . . 8
Where the configuration is searched . . . 8

Configuration **9**

The keys you must write . . . 9
`destinations`: the vault of each domain . . . 9
`destination`, `arborescence`, `executer_sur` . . . 10
`suivi`: state, journal and reports . . . 10
`ollama`: the models . . . 11
`extraction` and `classification` . . . 12
`categories`, `personnes`, `emetteurs_connus`, `pieces_communes` . . . 13
`nommage`: the name of a filed document . . . 13
`extensions_cibles` and `scan` . . . 14
`sources_locales` . . . 14
`ssh`, `sources_distantes`, `imap`, `oauth2`, `sources_imap` . . . 14
`mesure` and `inventaire` . . . 14

First pass **15**

Simulating: `--dry-run` . . . 15
Checking the routing without the models . . . 17
Filing: `--execute` . . . 18
Undoing: `--rollback` . . . 19
Exit codes . . . 20

Vault and encryption **21**

What a vault is . . . 21
The proof of mount . . . 21
An example with gocryptfs . . . 22
Where the state lives . . . 22
Space on the disk . . . 23

Sources over ssh **24**
What the remote machine must offer . . . 24
Declaring the machine . . . 24
A first run: list, then fetch . . . 25
Running it again . . . 26
When the connection fails . . . 27

IMAP mailboxes and Google accounts **28**
What is asked of the server . . . 28
Declaring a mailbox . . . 28
The password . . . 29
A first run: list, then deposit . . . 29
When the connection fails . . . 31
Google accounts with OAuth2 . . . 32

Inventory and rotation **34**
Reading what a vault already holds: `--inventaire` . . . 34
Keeping the journal short: rotation . . . 35

Diagnosis **36**
The states . . . 36
The checks, in order . . . 37
Reading a failure . . . 38
A fresh installation . . . 39
In another language . . . 39

Sovereignty and limits **40**
What the tool guarantees . . . 40
Respect of the household . . . 40
The limits . . . 40
Reporting a problem . . . 41

Introduction

Dibab (“to choose” in Breton) collects, analyses and files the personal administrative documents of a household: invoices, letters, contracts, notices from public bodies. It moves each document into a normalised tree, under a name that tells the date, the sender and the category, and it keeps a journal of everything it does so that any filing can be undone. This manual describes version 0.9.0-alpha.

It runs in sovereign mode. No data leaves your local network: there is no third-party service, no remote model and no cloud. The language models are those of Ollama, which runs on your own machine.

What Dibab does

- It scans local directories, remote machines over `ssh` and mailboxes over IMAP. A remote file is copied and checked by fingerprint, and the original is left intact. A mailbox is read without being modified: only the attachments are fetched, and no message is marked as read.
- It extracts the text of each file (the text layer of a PDF, otherwise a reading of the page images by a vision model), sets aside what is not administrative, then files the rest by category and by domain, personal or professional, with the agreement of several models.
- It renames and moves each document into the vault of its domain, writes a timestamped journal, and can undo a whole session with `--rollback`.

What it does not do

- It does not move a file it is not sure about. Doubt leads to a folder named `TO_CHECK/`, never to a risky filing.
- It never deletes a document. It copies, checks, then removes the original only once the copy is proven identical.
- It does not modify a mailbox or a remote machine.
- It does not run in the background. You start it by hand, and its default mode is the simulation, `--dry-run`.
- It does not mount, encrypt or synchronise a vault. That is up to you, and chapter 5 explains how the tool checks it.

A tool for a household

Dibab is designed for the administrative life of a household, by the household. A machine, a directory or a mail account enters its scope only with the agreement, explicit or implicit, of the person it concerns; when in doubt, leave it out. The software cannot guarantee this respect: the administrator of the instance commits to it.

The journal holds file names and senders, which already reveal a lot (a letter from a health body is enough). It therefore belongs in the vault, not beside it: put the `suivi.*` keys inside the vault, otherwise it is written in clear under `~/.local/state/dibab/`.

How this manual is organised

1. Introduction (this chapter).
2. Installation: what to install, where everything goes, and a first check.
3. Configuration: the configuration file, key by key.
4. First pass: simulate, read the report, file for real, undo.
5. Vault and encryption: the proof that a vault is mounted, and where to keep the state.
6. Sources over ssh: remote machines.
7. IMAP mailboxes and Google accounts.
8. Inventory and rotation: reading what a vault already holds, and keeping the journal short.
9. Diagnosis: `--diagnose`, state by state.
10. Sovereignty and limits: what the tool guarantees, and what it does not.

Conventions

Commands are shown as they are typed, in a terminal, for the user of the installation described in chapter 2. The command is `dibab`. The text that follows a command is its real output, produced by replaying the command on a trial area made of fictitious documents; the models named in those outputs (`essai-ocr`, `essai-tri`, `essai-classification`) are stand-ins of that trial area, and yours are the ones you name in your configuration. Every document, person, sender and company in this manual is invented.

The language of the messages follows your session: Dibab reads the first non-empty variable among `LC_ALL`, `LC_MESSAGES` and `LANG`, and speaks English, French, German, Spanish or Simplified Chinese accordingly (any other value gives English). The option names, the state tokens such as `OK` and `ÉCHEC`, the identifiers, the journals and the plan files never change with the language, so that a journal written in one language is replayed by `--rollback` in any other. The outputs of this manual are in English.

Installation

What you need

- A Linux system with Python 3.10 or later. Dibab uses only the Python standard library, so nothing else is installed for Python.
- `findmnt` (package `util-linux`), which proves that a vault is mounted.
- `pdftotext` and `pdftoppm` (package `poppler-utils`), which read PDF files.
- `ssh` (package `openssh-client`), only if you scan remote machines, with a key that opens a session without a password.
- Ollama, installed and running, with at least one model for each role (below). Dibab never installs Ollama and never downloads a model.
- A destination directory for each domain, personal and professional, preferably on an encrypted volume (chapter 5).
- `git`, to fetch the sources.

On Debian or Ubuntu, the packages are installed in one command:

```
sudo apt install python3 git util-linux poppler-utils openssh-client
```

The models, by role

Three roles use a model each. The role `ocr` reads scanned pages, `tri_rapide` ("quick triage") sets aside what is not administrative, and `classification` files. The models of the `classification` role vote: a document is filed only if their agreement reaches `ollama.seuil_confiance` (chapter 3). Two optional roles, `tri_decision` and `objet`, let a `decision` model do the triage in one pass and a small model give the subject; without them, `tri_rapide` sorts (chapter 3).

```
ollama pull glm-ocr
ollama pull mistral-nemo:12b-instruct-2407-q6_K
ollama pull gemma4
ollama pull gemma3:4b
ollama pull qwen2.5
ollama pull granite4.2:8b-Q5_K_M
```

These are the models of the reference instance; any others will do, as long as you name them in `ollama.roles`. List what is installed with `ollama list`.

The final tree

~ is your home directory. Adapt the paths to your installation.

```
~/applications/dibab/              the cloned repository
├── dibab.py                       the entry point
├── modules/                            one file per responsibility
│   └── langues/                        the message dictionaries: en, fr, de, es, zh
├── docs/                               the manual page and the translated README files
├── tests/                              the tests
```

```
└── config.exemple.json                the example configuration
~/.local/bin/dibab                 a link to dibab.py: the command
~/.config/dibab/config.json        your configuration, never versioned
~/.local/state/dibab/              the state, the journal and the session reports (default pla
 ↪  ce)
~/Classement/Personnel/                 the vault of the personal domain
~/Classement/Professionnel/             the vault of the professional domain
```

Installation from the Debian package

On Debian 12 and 13 and on Ubuntu 22.04 and 24.04, one file installs the program and its dependencies (`python3`, `poppler-utils`, `util-linux`, `openssh-client`). It contains neither Ollama, nor a model, nor a configuration. Download `dibab_0.9.0-alpha_all.deb` from the download page of dibab.davalan.fr, then, from the folder that holds it:

```
sudo apt install ./dibab_0.9.0-alpha_all.deb
```

Write the configuration, in the language of your session, then follow steps 4 and 5 of the installation from the sources below (the vault of each domain, the keys of the configuration):

```
dibab --initialiser
```

`--initialiser` writes the configuration in the language of your session: the check folder, its prefix and the duplicate folder take the names of that language. It creates the file with mode `600` in a folder with mode `700` and never rewrites a file that already exists; a later session in another language does not rename them.

Check the installation, without writing anything:

```
dibab --version
dibab --diagnose
```

The package puts the command in `/usr/bin/dibab`, the program in `/usr/lib/dibab/`, and the README files, the example configuration and the manual page (`man dibab`, in five languages) under `/usr/share/`. Removing it keeps your data: `sudo apt remove dibab` and `sudo apt purge dibab` delete the files of the package and never touch your configuration, your state, your journal or your documents; the purge names what it leaves.

Level: installation and diagnostic validated in a container on Debian 12 and 13 and on Ubuntu 22.04 and 24.04 (installation, `--version`, `--diagnose`, the refusal to write without a configuration and the refusal to write into a vault that is not mounted, a simulated filing, removal and purge). A container shows neither the mounting of an encrypted vault nor a run with your own Ollama and your own documents.

Installation on Fedora, openSUSE Leap and Arch

On Fedora 44, openSUSE Leap 16.0 and Arch Linux, one file also installs the program and its dependencies. Download the file of your system from the download page of dibab.davalan.fr, then, from the folder that holds it, run the line of your system:

```
sudo dnf install ./dibab-*.fc44.noarch.rpm
sudo zypper install --allow-unsigned-rpm ./dibab-*.leap160.noarch.rpm
sudo pacman -U ./dibab-*-any.pkg.tar.zst
```

The packages are not signed. The rest is the same as with the Debian package: `dibab --initialiser`, then `dibab --version` and `dibab --diagnose`. Removing the package (`sudo dnf remove dibab`, `sudo zypper remove dibab`, `sudo pacman -R dibab`) never touches your configuration, your state, your journal or your documents.

Level: installation and diagnostic validated in a container on Fedora 44, openSUSE Leap 16.0 and Arch Linux (installation, `--version`, `--diagnose`, the two refusals to write, a simulated filing, reinstallation over a broken installation, removal). A container shows neither the mounting of an encrypted vault nor a run with your own Ollama and your own documents.

Installing from the sources

1. Fetch the sources:

```
mkdir -p ~/applications
git clone https://github.com/RonanDavalan/dibab.git ~/applications/dibab
```

2. Make the command available. The entry point is executable; a link in `~/.local/bin` gives it the name `dibab` (on most distributions this directory is already in the `PATH` of a login shell):

```
mkdir -p ~/.local/bin
ln -s ~/applications/dibab/dibab.py ~/.local/bin/dibab
```

3. Write the configuration in the language of your session (chapter 3 explains each key):

```
dibab --initialiser
```

`--initialiser` writes the configuration in the language of your session: the check folder, its prefix and the duplicate folder take the names of that language. It creates the file with mode `600` in a folder with mode `700` and never rewrites a file that already exists; a later session in another language does not rename them.

4. Create the vault of each domain and read the type of its file system (chapter 5 explains why):

```
mkdir -p ~/Classement/Personnel ~/Classement/Professionnel
findmnt -n -T ~/Classement/Personnel -o FSTYPE
```

```
ext4
```

The command prints the type of the volume that holds the directory, for instance `ext4`. Write this type in `destinations.<domain>.systemes_fichiers_acceptes` for an ordinary directory, or open your encrypted volume first and write the type that `findmnt` reports for it.

5. Edit `~/.config/dibab/config.json`, at least the keys of chapter 3 marked as required, then check the installation without writing anything:

```
dibab --version
```

```
dibab 0.9.0-alpha
```

The command prints one line and stops. It reads no configuration, touches no file and talks to no service.

Running the tests (optional)

The sources carry their own tests, which need `pytest` in a virtual environment. They are not needed to use Dibab; run them after an update, or if you suspect the installation:

```
cd ~/applications/dibab && \
  python3 -m venv venv && \
  venv/bin/pip install -r requirements-dev.txt && \
  venv/bin/python -m pytest -q tests
```

The last line reports the passed tests and the skipped ones; the skipped ones are the trials against a running Ollama, which the test suite does not require.

Where the configuration is searched

Dibab reads the first file found among, in this order: the `--config <path>` option, the `DIBAB_CONFIG` environment variable, `$XDG_CONFIG_HOME/dibab/config.json` (`~/.config/dibab/config.json` when the variable is empty), then `config.json` next to `dibab.py`. There is no merging: the first file found is the only one read. An option or a variable that names a missing file stops the program with exit code 1. It never falls back on another file, because that would make it work on a configuration you did not designate, that is, on another vault.

Configuration

Nothing is hard-coded in Dibab. Every path, model name, threshold, delay and naming rule lives in one JSON file, `config.json`. Changing the behaviour of the tool means changing this file, not the code. The sources carry a complete, realistic example, `config.exemple.json`; this chapter explains it block by block. Chapter 2 says where the program looks for the file.

Paths written with ~ or `$HOME` are expanded. The example contains no absolute path. No secret is ever written in this file: a mail password goes in an environment variable, a Google token in a separate file with mode `600` (chapter 7).

A missing key is not silently replaced by a guess. The program stops with exit code 1 and names the missing keys, so `dibab --diagnose` (chapter 9) is the first thing to run after an edit.

The keys you must write

Open the file that `dibab --initialiser` wrote, then change at least these:

Key	What to write
`destinations.personnel.racine`, `destinations.professionnel.racine`	the root of the vault of each domain
`destinations.*.systemes_fichiers_acceptes`	the file system type that proves the vault mounted (chapter 5)
`personnes`	the first names of the household; they name the folders of the personal domain
`sources_locales[].chemin`	the directories to scan
`ollama.roles`	for each role, the preferred models, in order (`config.exemple.json` leaves the three lists empty)
`suivi.fichier_date`, `suivi.journal`, `suivi.sessions`	optional, but recommended inside the vault (chapter 5)

Delete the example entries of `sources_distantes` and `sources_imap` that you do not use: a source that is declared is a source that is read.

`destinations`: the vault of each domain

One entry per domain, `personnel` and `professionnel`.

Key	Meaning
`racine`	the root of the vault of this domain
`systemes_fichiers_acceptes`	the file system types that prove the vault mounted; the example expects `fuse.gocryptfs`
`sous_repertoire_classement`	the folder, under the root, where documents of this domain are filed (`Administratif` for the personal domain, `Classement` for the professional one)

Key	Meaning
sous_repertoire_a_verifier	the folder for the documents Dibab is not sure about (TO_CHECK/)

The name of the check folder and its prefix are written at initialisation in the language of the session: sous_repertoire_a_verifier sets the folder and nommage.prefixe_a_verifier the prefix. The folder at the root of a vault that receives duplicates is set by destination.repertoire_doublon.

destination, arborescence, executer_sur

Key	Meaning
destination.sous_repertoire_logs	the folder of the business logs inside a vault (logs); the inventory plans are written there
destination.vide_poches_prefixe	the prefix of the "drop folders" (vide-poches-), where fetched files wait
destination.repertoire_doublon	the folder, at the root of a vault, that receives a document whose fingerprint is already filed (duplicate/)
arborescence.personnel	the sub-path of a personal document under its filing folder: {personne}/{annee} gives Alice/2026
arborescence.professionnel	the same for a professional one: {annee}/{categorie} gives 2026/Factures
arborescence.personne_inconnue	the folder used when the person is not in personnes (Commun)
executer_sur	the machine names (hostname) where --execute is allowed; an empty list means any machine, and --execute elsewhere stops with exit code 4

suivi: state, journal and reports

Key	Meaning
fichier_date	the file dernier_scan.json, which remembers the date of the last scan of each source
journal	the action journal journal.jsonl, one JSON line per action, read back by --rollback; it carries paths, never document text
sessions	the folder of the session reports
logs_systeme	reserved for a technical log; nothing writes it today
securite_jours	how many days before the last scan a new scan restarts (1), so that a file added in parallel is not missed
premier_scan_jours	how far back the very first scan of a source goes (90 days)
rotation.journal_taille_max_mo	the size beyond which the journal is rotated (20 MB)
rotation.journal_archives_max	how many rotated journals are kept (5)

Key	Meaning
rotation.sessions_max	how many session files are kept (100)

The four path keys (fichier_date, journal, sessions, logs_systeme) are optional. Without them the files go under $XDG_STATE_HOME/dibab/, that is ~/.local/state/dibab/, as dernier_scan.json, journal.jsonl, sessions/ and dibab.log, with mode 600 for files and 700 for folders. These defaults are in clear text on the disk: chapter 5 explains how to move them into a vault. Three more files follow fichier_date wherever it is: rapatries.json, imap_deposes.json and dibab.verrou.

ollama: the models

Key	Meaning
url	the address of Ollama (http://localhost:11434); every call is a plain HTTP request to it
roles.ocr	the ordered preferences of the model that reads scanned pages
roles.tri_rapide	the ordered preferences of the model that sets aside what is not administrative
roles.classification	the models that file; they all run on each document and vote
roles.tri_decision	optional: the ordered preferences of a decision model (ollama show lists the capability) that sorts in one pass, with thresholds, instead of tri_rapide
roles.objet	optional: the small generation model that gives the subject of a document when the decision model sorts it
seuil_confiance	between 0 and 1; a domain is kept only if the share of classification models that agree reaches it (0.4 in the example)
timeout_secondes	the delay of each request to Ollama (120)
keep_alive_entre_pages	how long the OCR model stays loaded between two pages of a document (2m)

At start, Dibab asks Ollama for the installed models and keeps, for ocr and tri_rapide, the first preference that is installed; a name without a tag stands for :latest. For classification it keeps every installed preference, and they vote. A role with no installed model stops the program with exit code 3, lists the models present and names the line to correct, before anything is written. The two optional roles never stop it: when tri_decision is declared but unusable (no model installed, no decision capability, no thresholds, no model for objet, or an Ollama that does not know /v1/systemone), the triage goes through tri_rapide and the session header says why.

extraction and classification

Key	Meaning
extraction.seuil_caracteres_alphanumeriques	below this count of letters and digits, a PDF is read by the OCR model instead of its text layer (200)
extraction.pages_ocr	how many pages the OCR reads (3)
extraction.resolution_ocr	the resolution of the rendered pages, in dpi (200)
extraction.taille_max_texte	the extracted text is truncated to this many characters (20000)
extraction.repertoire_temporaire	where pages are rendered; keep it in memory (/dev/shm)
extraction.timeout_outils_secondes	the delay of pdftotext and pdftoppm (60)
extraction.consigne_ocr	the instruction sent to the OCR model with each image
classification.temperature, classification.num_ctx	the options sent with each request (0 and 8192); a context too short cuts the beginning of the prompt
classification.caracteres_max_prompt	how much of the text is sent to the classification models (12000)
classification.caracteres_max_tri	how much of the text is sent to the triage model (3000); the start of a document says what it is
classification.keep_alive_entre_documents	how long a model stays loaded between two documents (2m)
classification.longueur_max_champ, longueur_max_citation	the maximum length of the sender, the recipient and the subject kept, and of a quoted extract (80 and 120)
classification.marqueurs_mentions	words whose presence is a weak professional signal (TVA, SIRET, …)
classification.seuils_tri_decision	optional: for each decision model, non_administratif and administratif (between 0 and 1) and caracteres_min; the probability of not administrative given by the model means not administrative from the first, administrative below the second, uncertain in between, and a text shorter than caracteres_min letters and digits is uncertain. Thresholds hold for one model and one caracteres_max_tri: measure them again when either changes

The text of a document is treated as data, never as an instruction: it is delimited by a random marker, the model answers a closed JSON that Dibab validates, and any invalid answer counts as “uncertain”. A model can never trigger an action.

categories, personnes, emetteurs_connus, pieces_communes

Key	Meaning
categories	the closed list of categories (Factures, Aides, Impôts, ...); a model may answer only with one of them, or "none"
personnes	the household members; a model may answer only with one of them, or "none"
emetteurs_connus	known senders by category, each {"nom": ..., "domaine": "personnel" | "professionnel" | "mixte"}; a known sender is a signal of domain, and an empty list gives no such signal
pieces_communes	the categories copied into the vault of each domain, with the same name and the same fingerprint (Identité, Domicile, RIB), only if every classification model returns that same category

The categories and the known senders of the example are generic; adapt them to your country.

nommage: the name of a filed document

Key	Meaning
format	the template of a filed document, with {date}, {emetteur}, {categorie}, {personne}, {objet} (the subject returned by the triage), {ext} ({date}_{emetteur}_{categorie}{ext})
separateur, espaces_vers	the separator and the replacement of spaces (_ and -)
date_format	the date format (%Y-%m-%d)
date_inconnue_prefixe	the prefix of the modification date used when no date is found in the document (~)
emetteur_inconnu	the name used when the sender is unknown (Inconnu)
prefixe_a_verifier	the prefix of a document placed in the check folder ([TO_CHECK])
format_a_verifier	the template that follows the prefix, with the same variables and {origine}, the original name; an empty or unknown part is left out with its separator, and the whole name keeps to 180 characters ({date}_{emetteur}_{objet}_{origine}{ext}); without this key, the document keeps its original name
suffixe_collision	the suffix of a name already taken, from 2 (_{n})

With the configuration that `dibab --initialiser` writes, a notice from a public body dated 15 March 2026 becomes `2026-03-15_Organisme-public_Aides.pdf`, and a letter without a header placed in the check folder becomes `[TO_CHECK]courrier-de-suivi-de-dossier_courrier-sans-en-tete.pdf`.

extensions_cibles and scan

Key	Meaning
extensions_cibles	the extensions that are taken (.pdf, .png, .jpg, .jpeg), compared without regard to case
scan.recursif_par_defaut	scan sub-folders (true); each source can override it
scan.exclusions_automatiques	sub-folders never scanned, for every source (rebut, and the folder named by destination.repertoire_doublon)
scan.suivre_symlinks	follow symbolic links (false, so that the scan cannot leave its perimeter)
scan.ignorer_dotfiles	ignore hidden files and folders (true)

sources_locales

Each entry is a directory to scan.

Key	Meaning
id	a unique name; it is the key of the source in dernier_scan.json
chemin	the directory
date_reference	modification (recommended) or creation; the date used to keep recent files
recursif	overrides scan.recursif_par_defaut for this source
exclusions	sub-folder names to skip for this source only

The first scan of a source looks back suivi.premier_scan_jours days; the next ones restart from the date of the last complete scan, minus suivi.securite_jours.

ssh, sources_distantes, imap, oauth2, sources_imap

These blocks are explained where they are used: ssh and sources_distantes in chapter 6, imap, oauth2 and sources_imap in chapter 7.

mesure and inventaire

Key	Meaning
inventaire.exclusions	the folder names that --inventaire does not read (chapter 8)
mesure.graine, documents_par_coffre, exclusions, taux_erreur_max	read only by the measurement tool of the sources (_outils/mesurer_domaines.py), which samples filed documents and checks the models against the vault; the example values are a starting point

First pass

A first pass has four moments: put the documents in a source directory, simulate with `--dry-run`, read what the simulation proposes, then file for real with `--execute`. Every filing can be undone with `--rollback`. Always start with the simulation: it moves nothing, writes nothing in any vault, and shows exactly where each document would go.

This chapter uses the source directory `~/Classement/Entrant` of the configuration of the trial area, which holds four fictitious documents: a notice from a public body, a hosting invoice, a cake recipe and a letter without any header.

```
ls ~/Classement/Entrant

courrier-sans-en-tete.pdf
facture-hebergement.pdf
notification-organisme.pdf
recette-gateau.pdf
```

Simulating: `--dry-run`

`--dry-run` is the default mode: `dibab` alone does the same. It needs Ollama, because the whole classification runs, but it moves no file and writes neither `dernier_scan.json` nor any filed document.

```
dibab --dry-run

Dibab 0.9.0-alpha — session 2026-11-23T12:09:03
  Mode               : dry-run
  Configuration      : ~/.config/dibab/config.json
  Vault personnel     : ~/Classement/Personnel — mounted
  Vault professionnel : ~/Classement/Professionnel — mounted
  Session journal    : ~/.local/state/dibab/sessions/session_2026-11-23T12:09:03.jsonl
  Model ocr           : essai-ocr:latest
  Model tri_rapide    : essai-tri:latest
  Model classification: essai-classification:latest
  Consensus          : 1 classification model(s), threshold 0.4

Source: entrant (~/Classement/Entrant)
  - notification-organisme.pdf                        (928 B, mtime 2026-11-23)
  - facture-hebergement.pdf                           (922 B, mtime 2026-11-23)
  - recette-gateau.pdf                                (895 B, mtime 2026-11-23)
  - courrier-sans-en-tete.pdf                         (901 B, mtime 2026-11-23)
  Source total: 4 file(s), 3.6 KB

============================================================
Scan total: 4 candidate file(s), 3.6 KB
============================================================
Text extraction (the text is neither shown nor written)
  - notification-organisme.pdf: text layer, 258 character(s), 1 page(s), 0.02 s
  - facture-hebergement.pdf: text layer, 246 character(s), 1 page(s), 0.02 s
  - recette-gateau.pdf: text layer, 226 character(s), 1 page(s), 0.02 s
```

```
  - courrier-sans-en-tete.pdf: text layer, 236 character(s), 1 page(s), 0.02 s

Classification (dry-run, nothing is moved; the text is neither shown nor written)
model 1/1, document 1/3
model 1/1, document 2/3
model 1/1, document 3/3
  - notification-organisme.pdf: triage administrative, domain personnel, agreement 1/1
      subject: notification de paiement d'allocation
      -> vault personnel, classement: ~/Classement/Personnel/Administratif/Alice/2026/2026-03-1
       ↪  5_Organisme-public_Aides.pdf
  - facture-hebergement.pdf: triage administrative, domain professionnel, agreement 1/1
      subject: facture d'hébergement
      -> vault professionnel, classement: ~/Classement/Professionnel/Classement/2026/Factures/2
       ↪  026-03-15_Hebergeur-Exemple_Factures.pdf
  - recette-gateau.pdf: triage not administrative, domain not applicable
      subject: recette de gâteau
      -> stays in place (not administrative)
  - courrier-sans-en-tete.pdf: triage administrative, domain incertain, agreement 1/1
      subject: courrier de suivi de dossier
      reason: no recipient quoted and no known issuer
      -> vault personnel, a_verifier: ~/Classement/Personnel/TO_CHECK/[TO_CHECK]courrier-de-sui
       ↪  vi-de-dossier_courrier-sans-en-tete.pdf (not moved)
```

The progress lines (`model 1/1, document 1/3`) go to the error output, one per model and per document. The first pass is slow, since every classification model reads every document, and these lines tell you the program is not frozen.

Reading the report

The report has four parts, in this order.

1. The header: the version and the time of the session, the mode, the configuration file that was read, each vault with the proof that it is mounted, the path of the session journal, the model kept for each role and the consensus rule.
2. The scan: for each source, the candidate files with their size and date, then the totals. A file is a candidate when its extension is in `extensions_cibles` and its date is recent enough.
3. The text extraction: for each candidate, how its text was obtained (`text layer`, or `OCR` with the model that read it), how many characters and pages, and how long it took. The text itself is neither shown nor written.
4. The classification: one block per document. The first line gives the triage (administrative, not administrative or uncertain), the domain that Dibab retains and the agreement, which counts the classification models that return that domain among those asked (`agreement 2/3`: two models out of three). The `subject:` line says in a few words what the document is, as the triage model read it; it is never an extract of the text. For a document sent to the check folder, the `reason:` line says why, from a closed list. The last line, after `->`, gives the destination.

In execution, a last line names the session report.

A destination reads `-> vault <domain>, <zone>: <path>`. The zone tells what will happen:

Zone or text	Meaning
`classement`	the document is filed in the classified zone of that vault, under its normalised name
`a_verifier`	Dibab is not sure: the document goes to the check folder, `TO_CHECK/`, under the prefix `[TO_CHECK]` followed by its date, sender and subject when the document gives them, and by its original name; the simulation adds `(not moved)` because nothing is filed in the classified zone
`stays in place (not administrative)`	the triage set the document aside; it is neither moved nor renamed, and it is listed again at every pass
`(duplicate: would go to duplicate/)`	the fingerprint of the document is already filed; it goes to the `duplicate/` folder instead of being filed twice

The domain `incertain` always leads to the check folder, and so does a document on which the models do not agree. A common piece (an identity document, a proof of address, a bank identity) is proposed in the vault of each domain only when every classification model returns the same category; otherwise it goes to the check folder. In doubt, Dibab prefers a human decision to a risky filing.

Checking the routing without the models

To see which destination a verdict leads to without waiting for any model, give `--dry-run` a JSON file that associates a file name with a verdict. The routing table is applied to it and the destination is displayed. The option exists only with `--dry-run`: the program refuses it, with exit code 2, together with `--execute`.

```
cat > verdicts.json <<'EOF'
{
  "notification-organisme.pdf": {"administratif": true, "domaine": "personnel", "categorie": "A
   ↪  ides", "personne": "Alice", "date": "2026-03-15", "emetteur": "Organisme public"},
  "recette-gateau.pdf": {"administratif": false, "categorie": "Inconnue"}
}
EOF
dibab --dry-run --verdicts-simules verdicts.json | sed -n '/Simulated routing/,$p'

model 1/1, document 1/3
model 1/1, document 2/3
model 1/1, document 3/3
Simulated routing (verdicts supplied, nothing is moved)
  - notification-organisme.pdf: domain personnel
      -> vault personnel, classement: ~/Classement/Personnel/Administratif/Alice/2026/2026-03-1
       ↪  5_Organisme-public_Aides.pdf
  - facture-hebergement.pdf: no simulated verdict
  - recette-gateau.pdf: not administrative, stays in place
  - courrier-sans-en-tete.pdf: no simulated verdict
```

Filing: `--execute`

When the simulation suits you, run the same command with `--execute`. Each document is copied to its destination, the copy is checked by its SHA256 fingerprint, and only then is the original removed from the source. Nothing is removed before the copy is proven identical.

```
dibab --execute

Dibab 0.9.0-alpha — session 2026-11-23T12:09:03
  Mode               : execute
  Configuration      : ~/.config/dibab/config.json
  Vault personnel     : ~/Classement/Personnel — mounted
  Vault professionnel : ~/Classement/Professionnel — mounted
  Session journal    : ~/.local/state/dibab/sessions/session_2026-11-23T12:09:03.jsonl
  Model ocr           : essai-ocr:latest
  Model tri_rapide    : essai-tri:latest
  Model classification: essai-classification:latest
  Consensus          : 1 classification model(s), threshold 0.4

Source: entrant (~/Classement/Entrant)
  - notification-organisme.pdf                          (928 B, mtime 2026-11-23)
  - facture-hebergement.pdf                             (922 B, mtime 2026-11-23)
  - recette-gateau.pdf                                  (895 B, mtime 2026-11-23)
  - courrier-sans-en-tete.pdf                           (901 B, mtime 2026-11-23)
  Source total: 4 file(s), 3.6 KB

============================================================
Scan total: 4 candidate file(s), 3.6 KB
============================================================
Text extraction (the text is neither shown nor written)
  - notification-organisme.pdf: text layer, 258 character(s), 1 page(s), 0.02 s
  - facture-hebergement.pdf: text layer, 246 character(s), 1 page(s), 0.02 s
  - recette-gateau.pdf: text layer, 226 character(s), 1 page(s), 0.02 s
  - courrier-sans-en-tete.pdf: text layer, 236 character(s), 1 page(s), 0.02 s

Filing (execution: each document is copied, checked by SHA256, then removed from its source)
model 1/1, document 1/3
model 1/1, document 2/3
model 1/1, document 3/3
  - notification-organisme.pdf: classe
      -> ~/Classement/Personnel/Administratif/Alice/2026/2026-03-15_Organisme-public_Aides.pdf
  - facture-hebergement.pdf: classe
      -> ~/Classement/Professionnel/Classement/2026/Factures/2026-03-15_Hebergeur-Exemple_Factu
       ↪  res.pdf
  - recette-gateau.pdf: not administrative, stays in place
  - courrier-sans-en-tete.pdf: a_verifier
      -> ~/Classement/Personnel/TO_CHECK/[TO_CHECK]courrier-de-suivi-de-dossier_courrier-sans-e
       ↪  n-tete.pdf

Session report: ~/.local/state/dibab/sessions/rapport_2026-11-23T12:09:03.json
```

In execution, each line ends with the action performed: `classe` (filed in the classified zone), `a_verifier` (placed in the check folder) or `doublon` (placed in the `duplicate/` folder). These words are the same in every language, like the actions written in the journal.

The documents are now where the report said:

```
find ~/Classement -type f | sort
```

```
~/Classement/Entrant/recette-gateau.pdf
~/Classement/Personnel/Administratif/Alice/2026/2026-03-15_Organisme-public_Aides.pdf
~/Classement/Personnel/TO_CHECK/[TO_CHECK]courrier-de-suivi-de-dossier_courrier-sans-en-tete.pdf
~/Classement/Professionnel/Classement/2026/Factures/2026-03-15_Hebergeur-Exemple_Factures.pdf
```

Three files left `Entrant`. The recipe stayed: it is not administrative, and Dibab does not touch what it was not asked to file.

What `--execute` writes

Every move is written in the action journal, `suivi.journal`, one JSON line per action:

```
head -n 1 ~/.local/state/dibab/journal.jsonl
```

```
{"schema_version": 1, "session": "2026-11-23T12:09:03", "timestamp": "2026-11-23T12:09:03",
 ↪  "evenement": "action", "id_action": "<identifiant>", "action": "classe", "domaine": "person
 ↪  nel", "origine": "~/Classement/Entrant/notification-organisme.pdf", "empreinte_avant":
 ↪  "6850987e722babfbca9f474ce8971afc532762ac07cd767308c8294fc1383d28", "destinations":
 ↪  [{"chemin": "~/Classement/Personnel/Administratif/Alice/2026/2026-03-15_Organisme-public_Ai
 ↪  des.pdf", "empreinte_apres":
 ↪  "6850987e722babfbca9f474ce8971afc532762ac07cd767308c8294fc1383d28"}]}
```

The line holds the action and its identifier, the domain, the paths and the fingerprints before and after, and no text of the document. Every session also leaves two files in the folder `suivi.sessions`: a session journal (`session_<time>.jsonl`, the same events as the screen) and a report (`rapport_<time>.json`) with the count of each action, the failures and the duration, and no document name. The date of the scan of each source advances in `dernier_scan.json` only for the sources that were read without failure. Chapter 8 explains how these files are kept short.

Only one execution runs at a time. A second `--execute` or `--rollback` that finds the lock `dibab.verrou` taken stops with exit code 5 and touches nothing.

Undoing: `--rollback`

`--rollback` undoes the last session of filing. `--rollback --session <time>` undoes a given session, whose time is the one printed in its header and used in the names of its files.

```
dibab --rollback
```

```
Dibab 0.9.0-alpha — session 2026-11-23T12:09:03
  Mode               : rollback
  Configuration      : ~/.config/dibab/config.json
  Vault personnel     : ~/Classement/Personnel — mounted
```

```
  Vault professionnel : ~/Classement/Professionnel — mounted
  Session journal     : ~/.local/state/dibab/sessions/session_2026-11-23T12:09:03.jsonl

Rollback of session 2026-11-23T12:09:03: 3 undone, 0 failed
```

Each document returns to its original place, the copy checked by fingerprint, and its destination is removed. A line `annulation` is added to the journal, which is never rewritten. A common piece is removed from the vault of each domain together. An action that was already undone is not played twice.

```
find ~/Classement/Entrant -type f | sort

~/Classement/Entrant/courrier-sans-en-tete.pdf
~/Classement/Entrant/facture-hebergement.pdf
~/Classement/Entrant/notification-organisme.pdf
~/Classement/Entrant/recette-gateau.pdf
```

When an action cannot be undone, nothing is removed for it and the reason is displayed, with exit code 1. The reason is `destination modified since the filing` or `destination not found` when the document was edited or moved by hand (put it back or leave it), or `source occupied by another file` when another file took the original place (move it, then run the command again).

Exit codes

Code	Meaning
`0`	success
`1`	configuration not found, invalid or missing a key; at least one source or document in failure; rollback only partly done; OAuth2 consent failed; `--diagnose` with at least one `ÉCHEC`
`2`	vault not proven mounted before a mode that writes; invalid command line
`3`	no model installed for a role, or Ollama does not answer
`4`	`--execute` refused on this machine (key `executer_sur`)
`5`	another run holds the lock `dibab.verrou`

`--version` returns `0`, or `2` for an invalid command line.

Vault and encryption

What a vault is

A vault is the directory that receives the documents of one domain: one for the personal domain, one for the professional domain (`destinations.personnel.racine` and `destinations.professionnel.racine`). Dibab does not create, mount, encrypt or synchronise it. It expects the directory to be there, and, because the documents are sensitive, it expects it to sit on an encrypted volume that you open before the run.

The danger is easy to describe. When an encrypted volume is closed, its mount point is only an empty directory on the plain disk. A program that writes there writes the documents in clear text, below the encryption, without any error. Dibab therefore proves, before every write, that each vault is really mounted.

The proof of mount

Before `--execute`, `--rollback`, `--inventaire` or `--autoriser-google`, Dibab runs `findmnt -n -T <root> -o FSTYPE` for the root of each vault and compares the file system type with the list `systemes_fichiers_acceptes` of that vault. It also checks that the root is writable. If the type is not in the list, the vault is not proven and the program stops with exit code 2 without touching anything.

The example configuration expects `fuse.gocryptfs`, the type that `findmnt` reports for a mounted gocryptfs volume. With an ordinary directory in its place, which is what a closed vault looks like, the refusal is immediate:

```
dibab --execute --config ~/.config/dibab/config-ferme.json; echo "exit code: $?"

ERROR: vault personnel: vault not mounted (type ext4, expected fuse.gocryptfs) (~/Classement/Pe
↪  rsonnel)
ERROR: vault professionnel: vault not mounted (type ext4, expected fuse.gocryptfs) (~/Classemen
↪  t/Professionnel)
        Nothing was touched. Mount the destination directory, then run again.
exit code: 2
```

The simulation does not stop on a vault that is not proven: it signals it with `[!]` in the header and goes on, so that you can read the proposals before opening the vault. It does not write its session journal inside a path the vault contains, since that would put the journal in clear text on the disk.

Choosing the types to accept

Read the type of your own volume once it is open, with the command of chapter 2, and write it in the list:

```
"systemes_fichiers_acceptes": ["fuse.gocryptfs"]
```

If you encrypt differently (LUKS, eCryptfs, a volume of another tool), open the volume and write the type that `findmnt` returns for it. On an unencrypted volume the check can no longer tell an absent mount from a present one: you then give up this protection, and the diagnosis of chapter 9 reminds you that the state is in clear text.

An example with gocryptfs

gocryptfs is not part of Dibab, and nothing in Dibab requires it. This is one way to build the vault of each domain. Create each encrypted directory once, choosing a passphrase that you keep outside the machine; the command asks for it.

```
mkdir -p ~/Chiffre/Personnel ~/Chiffre/Professionnel
gocryptfs -init ~/Chiffre/Personnel
gocryptfs -init ~/Chiffre/Professionnel
```

Open each vault before a run. The command asks for the passphrase again and mounts the clear view on the directory named in the configuration:

```
gocryptfs ~/Chiffre/Personnel ~/Classement/Personnel
gocryptfs ~/Chiffre/Professionnel ~/Classement/Professionnel
```

Close them after the run:

```
fusermount3 -u ~/Classement/Personnel
fusermount3 -u ~/Classement/Professionnel
```

The encrypted directories `~/Chiffre/...` are what you back up and synchronise. Dibab knows nothing of them.

Where the state lives

Beside the documents, Dibab keeps a state: the date of the last scan of each source, the journal of actions, the session journals and reports, the register of fetched files and the lock. Without any `suivi.*` path in the configuration, all of it goes under `~/.local/state/dibab/` (or `$XDG_STATE_HOME/dibab/`), in clear text, with mode `700` for folders and `600` for files. The journal holds file names and senders; that is already revealing. The diagnosis flags it as a warning: `AVERTISSEMENT emplacement_etat`.

To encrypt the state, put it inside a vault, with three keys of the `suivi` block:

```
"suivi": {
  "fichier_date": "~/Classement/Personnel/logs/dernier_scan.json",
  "journal": "~/Classement/Personnel/logs/journal.jsonl",
  "sessions": "~/Classement/Personnel/logs/sessions"
}
```

The other files of the state (`rapatries.json`, `imap_deposes.json`, `dibab.verrou`) follow `fichier_date`. With this setting the diagnosis no longer reports a warning.

Space on the disk

Dibab moves and copies, and never deletes. A document that is filed is copied to its destination, checked, then removed from the source, so the disk space does not grow with a filing. A document that is a duplicate of one already filed goes to the `duplicate/` folder of the vault instead of being filed twice; emptying that folder is your decision, never the tool's.

Sources over ssh

A remote machine of the household is read over `ssh`. Dibab lists the files of the directories you name, copies them into a drop folder of the personal vault, checks each copy by its fingerprint, and then files them like local documents. The original files stay intact on the remote machine: Dibab never removes or modifies anything there.

What the remote machine must offer

- A Linux system with GNU `find` (the listing uses `-printf`), `sha256sum` and `cat`.
- An `ssh` alias that opens a session without a password. Everything about the connection, the address, the user and the key, lives in the `~/.ssh/config` of the local machine, under the alias; Dibab knows none of it. For example, with an invented machine:

```
Host machine1
    HostName 192.0.2.10
    User archive
    IdentityFile ~/.ssh/id_ed25519
```

Check the alias by hand before the first run. The command must print nothing and return `0`:

```
ssh machine1 true
```

Declaring the machine

Two blocks of `config.json` are involved: `ssh`, which gives the delays, and `sources_distantes`, which lists the machines.

```
"ssh": {
  "delai_connexion_secondes": 10,
  "delai_commande_secondes": 120,
  "delai_copie_secondes": 900
},

"sources_distantes": [
  {
    "id": "ssh-machine1",
    "alias_ssh": "machine1",
    "vide_poches_destination": "vide-poches-machine1",
    "repertoires": ["~/Téléchargements"],
    "date_reference": "modification",
    "recursif": true,
    "exclusions": []
  }
]
```

Key	Meaning
`ssh.delai_connexion_secondes`	the delay to open the connection (`10`)
`ssh.delai_commande_secondes`	the delay of a listing or of a fingerprint (`120`)

Key	Meaning
`ssh.delai_copie_secondes`	the delay to copy one file (`900`)
`id`	a unique name; it is the key of the machine in `dernier_scan.json`
`alias_ssh`	the alias of `~/.ssh/config`
`vide_poches_destination`	the name of the drop folder, without any separator, created under the root of the personal vault; the fetched files arrive there
`repertoires`	the directories to read on the remote machine; ~ is the home directory of the remote user
`date_reference`, `recursif`, `exclusions`	as for a local source (chapter 3)

The three `ssh` delays are required as soon as a remote source is declared.

A first run: list, then fetch

The configuration of this chapter declares the machine and nothing else. The simulation lists the files without copying anything:

```
dibab --dry-run --config ~/.config/dibab/config-ssh.json

Dibab 0.9.0-alpha — session 2026-11-23T12:09:03
  Mode               : dry-run
  Configuration      : ~/.config/dibab/config-ssh.json
  Vault personnel     : ~/Classement/Personnel — mounted
  Vault professionnel : ~/Classement/Professionnel — mounted
  Session journal    : ~/.local/state/dibab/sessions/session_2026-11-23T12:09:03.jsonl
  Model ocr           : essai-ocr:latest
  Model tri_rapide    : essai-tri:latest
  Model classification: essai-classification:latest
  Consensus          : 1 classification model(s), threshold 0.4

Remote source: ssh-machine1 (nothing is fetched in dry-run)
  - releve-bancaire.pdf                                (remote, 920 B, mtime 2026-11-23)

============================================================
Scan total: 0 candidate file(s), 0 B
============================================================
Text extraction (the text is neither shown nor written)

Classification (dry-run, nothing is moved; the text is neither shown nor written)
```

The block `Remote source` shows each file with the word `remote`. Nothing is written in the drop folder. On a first scan, only the files modified during the last `suivi.premier_scan_jours` days (90 by default) are listed, and only those whose extension is in `extensions_cibles`.

The execution fetches the files, then files them:

```
dibab --execute --config ~/.config/dibab/config-ssh.json

Dibab 0.9.0-alpha — session 2026-11-23T12:09:03
  Mode               : execute
  Configuration      : ~/.config/dibab/config-ssh.json
  Vault personnel     : ~/Classement/Personnel — mounted
  Vault professionnel : ~/Classement/Professionnel — mounted
  Session journal    : ~/.local/state/dibab/sessions/session_2026-11-23T12:09:03.jsonl
  Model ocr           : essai-ocr:latest
  Model tri_rapide    : essai-tri:latest
  Model classification: essai-classification:latest
  Consensus          : 1 classification model(s), threshold 0.4

Transfer of remote sources (copy checked by fingerprint, nothing is removed on the remote side)
  ssh-machine1: 1 candidate(s)

Source: ssh-machine1-vide-poches (~/Classement/Personnel/vide-poches-machine1)
  - releve-bancaire.pdf                              (920 B, mtime 2026-11-23)
  Source total: 1 file(s), 920 B

============================================================
Scan total: 1 candidate file(s), 920 B
============================================================
Text extraction (the text is neither shown nor written)
  - releve-bancaire.pdf: text layer, 242 character(s), 1 page(s), 0.02 s

Filing (execution: each document is copied, checked by SHA256, then removed from its source)
model 1/1, document 1/1
  - releve-bancaire.pdf: classe
      -> ~/Classement/Personnel/Administratif/Bob/2026/2026-03-15_Banque-Exemple_Banques.pdf

Session report: ~/.local/state/dibab/sessions/rapport_2026-11-23T12:09:03.json
```

Each file is copied into `vide-poches-machine1`, read back, and compared by SHA256 with the fingerprint computed on the remote machine. A copy that does not match is not kept. The scan then treats the drop folder as a local source, without any date filter, because a drop folder is emptied as the filing goes: whatever is in it is waiting. The register `rapatries.json`, next to `dernier_scan.json`, remembers the fetched files by fingerprint.

Running it again

A second execution fetches nothing new. The remote file is listed again, but its fingerprint is in the register, so it is neither copied nor filed a second time:

```
dibab --execute --config ~/.config/dibab/config-ssh.json

Dibab 0.9.0-alpha — session 2026-11-23T12:09:03
  Mode               : execute
  Configuration      : ~/.config/dibab/config-ssh.json
  Vault personnel     : ~/Classement/Personnel — mounted
  Vault professionnel : ~/Classement/Professionnel — mounted
```

```
  Session journal    : ~/.local/state/dibab/sessions/session_2026-11-23T12:09:03.jsonl
  Model ocr           : essai-ocr:latest
  Model tri_rapide    : essai-tri:latest
  Model classification: essai-classification:latest
  Consensus          : 1 classification model(s), threshold 0.4

Transfer of remote sources (copy checked by fingerprint, nothing is removed on the remote side)
  ssh-machine1: 1 candidate(s)

Source: ssh-machine1-vide-poches (~/Classement/Personnel/vide-poches-machine1)
  (no candidate file)
  Source total: 0 file(s), 0 B

============================================================
Scan total: 0 candidate file(s), 0 B
============================================================
Text extraction (the text is neither shown nor written)

Filing (execution: each document is copied, checked by SHA256, then removed from its source)

Session report: ~/.local/state/dibab/sessions/rapport_2026-11-23T12:09:03.json
```

The date of the machine in `dernier_scan.json` advances only after a complete listing in which every copy was verified. A file that failed is fetched again at the next execution, without duplicate. A file that exists both on the remote machine and in a local scan goes to the `duplicate/` folder: it is filed once.

When the connection fails

A refusal stops the machine concerned, and only that one; the other sources go on. Dibab does not retry in a loop, and neither should you.

Message	Likely cause
`Could not resolve hostname` or `Permission denied`	the machine is off, or the alias or the key is wrong; correct it, then run once more
a delay exceeded	raise `ssh.delai_commande_secondes` (listing) or `ssh.delai_copie_secondes` (copy)
`remote directory partly readable or absent`	correct `repertoires`
no candidate although the machine has files	the files are older than `suivi.premier_scan_jours`, or their extension is not in `extensions_cibles`

IMAP mailboxes and Google accounts

Dibab reads the attachments of a mailbox over IMAP, with a password or, for a Google account, with an OAuth2 token. It reads without modifying: the mailbox is opened read-only, no message is marked as read, no flag is set, and nothing is copied, moved or deleted on the server.

What is asked of the server

The program sends the server only `EXAMINE` (the read-only opening of a folder), `UID SEARCH ... SINCE` (the messages received since a date), and `UID FETCH` of the date of reception, of the structure of the message and of one attachment at a time with `BODY.PEEK`, the form that does not mark the message as read. The body of a message is never requested. The connection is always TLS, and the certificate is verified against the store of the system: a source with `"ssl": false` is refused.

An attachment is a part of the message that carries a name and whose disposition is `attachment`, or `inline` without a `Content-ID` (the images of a signature carry one and stay in the message). Its extension must be in `extensions_cibles`. Its name is data supplied by the sender: it is decoded and cleaned like the name of a fetched file (no separator, no control character, never hidden, bounded length), and it enters neither the session journal nor the report.

Declaring a mailbox

```
"imap": {
  "delai_secondes": 60,
  "taille_max_piece_mo": 25
},

"sources_imap": [
  {
    "id": "imap-foyer",
    "nom": "foyer",
    "serveur": "imap.example.com",
    "port": 993,
    "ssl": true,
    "authentification": "login",
    "utilisateur": "utilisateur@example.com",
    "mot_de_passe_env": "DIBAB_IMAP_FOYER",
    "dossiers": ["INBOX"],
    "vide_poches_destination": "vide-poches-foyer",
    "date_reference": "reception"
  }
]
```

Key	Meaning
`imap.delai_secondes`	the delay of each network exchange, connection included (`60`)
`imap.taille_max_piece_mo`	an attachment larger than this, according to the size the server announces, is not fetched (`25`)

Key	Meaning
`id`	a unique name; it is the key of the source in `dernier_scan.json`
`nom`	a label
`serveur`, `port`, `utilisateur`	the account
`ssl`	must be `true`
`authentification`	`login` (password in an environment variable) or `oauth2` (Google account); any other value stops the source
`mot_de_passe_env`	the name of the environment variable that holds the password (`login` only)
`dossiers`	the folders to read, each opened read-only
`vide_poches_destination`	the drop folder, a single name, under the root of the personal vault
`date_reference`	`reception`: the date of reception recorded by the server, never the `Date` header, which the sender writes

The two blocks `imap` and `sources_imap` are required together; a missing key stops the program with exit code 1 and names it.

The password

The password is never in `config.json`. Dibab reads it from the environment variable named by `mot_de_passe_env`, and never writes it, logs it or displays it. Define it in the terminal that starts Dibab, without writing it anywhere:

```
read -rs DIBAB_IMAP_FOYER && export DIBAB_IMAP_FOYER
```

A source whose variable is not defined is skipped with a warning, without any connection. That is also what happens to the example entry of `config.exemple.json` as long as you leave it declared.

A first run: list, then deposit

The configuration of this chapter declares the mailbox and nothing else. The simulation lists the attachments that would be fetched, with the word `imap`, and downloads none:

```
dibab --dry-run --config ~/.config/dibab/config-imap.json
```

```
Dibab 0.9.0-alpha — session 2026-11-23T12:09:03
  Mode               : dry-run
  Configuration      : ~/.config/dibab/config-imap.json
  Vault personnel     : ~/Classement/Personnel — mounted
  Vault professionnel : ~/Classement/Professionnel — mounted
  Session journal    : ~/.local/state/dibab/sessions/session_2026-11-23T12:09:03.jsonl
  Model ocr           : essai-ocr:latest
  Model tri_rapide    : essai-tri:latest
  Model classification: essai-classification:latest
  Consensus          : 1 classification model(s), threshold 0.4
```

```
IMAP source: imap-foyer (nothing is downloaded in dry-run)
  - attestation-assurance.pdf                          (imap, 1274 B, received 2026-11-23)

============================================================
Scan total: 0 candidate file(s), 0 B
============================================================
Text extraction (the text is neither shown nor written)

Classification (dry-run, nothing is moved; the text is neither shown nor written)
```

The execution downloads each attachment, checks its fingerprint, drops it in the drop folder and files it like a local document:

```
dibab --execute --config ~/.config/dibab/config-imap.json

Dibab 0.9.0-alpha — session 2026-11-23T12:09:03
  Mode               : execute
  Configuration      : ~/.config/dibab/config-imap.json
  Vault personnel     : ~/Classement/Personnel — mounted
  Vault professionnel : ~/Classement/Professionnel — mounted
  Session journal    : ~/.local/state/dibab/sessions/session_2026-11-23T12:09:03.jsonl
  Model ocr           : essai-ocr:latest
  Model tri_rapide    : essai-tri:latest
  Model classification: essai-classification:latest
  Consensus          : 1 classification model(s), threshold 0.4

IMAP sources (read-only: nothing is modified or deleted in the mailbox)
IMAP source: imap-foyer
  - attestation-assurance.pdf                          (imap, 1274 B, received 2026-11-23)

Source: imap-foyer-vide-poches (~/Classement/Personnel/vide-poches-foyer)
  - attestation-assurance.pdf                          (928 B, mtime 2026-11-23)
  Source total: 1 file(s), 928 B

============================================================
Scan total: 1 candidate file(s), 928 B
============================================================
Text extraction (the text is neither shown nor written)
  - attestation-assurance.pdf: text layer, 258 character(s), 1 page(s), 0.02 s

Filing (execution: each document is copied, checked by SHA256, then removed from its source)
model 1/1, document 1/1
  - attestation-assurance.pdf: classe
      -> ~/Classement/Personnel/Administratif/Alice/2026/2026-03-15_Assurance-Exemple_Assurance
       ↪  s.pdf

Session report: ~/.local/state/dibab/sessions/rapport_2026-11-23T12:09:03.json
```

The file `imap_deposes.json`, next to `dernier_scan.json`, remembers what was dropped (folder, `UIDVALIDITY`, `UID`, part and fingerprint). An attachment whose fingerprint is already in it, or in the action

journal, is not dropped again:

```
dibab --execute --config ~/.config/dibab/config-imap.json

Dibab 0.9.0-alpha — session 2026-11-23T12:09:03
  Mode               : execute
  Configuration      : ~/.config/dibab/config-imap.json
  Vault personnel     : ~/Classement/Personnel — mounted
  Vault professionnel : ~/Classement/Professionnel — mounted
  Session journal    : ~/.local/state/dibab/sessions/session_2026-11-23T12:09:03.jsonl
  Model ocr           : essai-ocr:latest
  Model tri_rapide    : essai-tri:latest
  Model classification: essai-classification:latest
  Consensus          : 1 classification model(s), threshold 0.4

IMAP sources (read-only: nothing is modified or deleted in the mailbox)
IMAP source: imap-foyer
  - attestation-assurance.pdf                          (imap, 1274 B, received 2026-11-23)

Source: imap-foyer-vide-poches (~/Classement/Personnel/vide-poches-foyer)
  (no candidate file)
  Source total: 0 file(s), 0 B

============================================================
Scan total: 0 candidate file(s), 0 B
============================================================
Text extraction (the text is neither shown nor written)

Filing (execution: each document is copied, checked by SHA256, then removed from its source)

Session report: ~/.local/state/dibab/sessions/rapport_2026-11-23T12:09:03.json
```

The date of the source in `dernier_scan.json` advances only after a complete reading in which every deposit succeeded.

When the connection fails

An authentication refusal or a failed connection stops the source, with no second attempt. The other sources go on.

Message	Likely cause
`environment variable ... not defined, source skipped`	export the password in the terminal that starts Dibab
`authentication refused by the server`	the password or the user is wrong; correct it, then run once more, not in a loop
`connection impossible`	the server, the port or the network
`folder ... not found`	correct `dossiers`
`attachment(s) above imap.taille_max_piece_mo`	raise this value

Google accounts with OAuth2

A Google account is read with an OAuth2 token instead of a password. The steps in the Google console are yours to do, with the Google account to read, and the labels of the console change over time: the notions are those of its OAuth2 documentation.

1. In the Google Cloud console, create a project, enable the Gmail API, and configure the OAuth consent screen (type “external”, your address as contact). Then publish the application, so that its status is “In production”. With the status “Testing”, the authorisation expires after seven days.
2. Create an OAuth credential of the type “Desktop application” and download its JSON file.
3. Keep that file in the vault, in a folder with mode `700`, with mode `600`. Dibab never modifies it. It is the `client_fichier`.
4. Declare the source with `"authentification": "oauth2"`, `client_fichier` and `token_fichier` instead of `mot_de_passe_env`, and add the block `oauth2`:

```
"oauth2": {
  "url_autorisation": "https://accounts.google.com/o/oauth2/v2/auth",
  "url_jeton": "https://oauth2.googleapis.com/token",
  "portee": "https://mail.google.com/",
  "marge_renouvellement_secondes": 300,
  "delai_secondes": 30,
  "delai_consentement_secondes": 300,
  "chemins_interdits": ["~/Documents"]
},

"sources_imap": [
  {
    "id": "imap-google",
    "nom": "foyer-google",
    "serveur": "imap.gmail.com",
    "port": 993,
    "ssl": true,
    "authentification": "oauth2",
    "utilisateur": "utilisateur@example.com",
    "client_fichier": "~/Classement/Personnel/logs/client-google.json",
    "token_fichier": "~/Classement/Personnel/logs/jeton-google.json",
    "dossiers": ["INBOX"],
    "vide_poches_destination": "vide-poches-google",
    "date_reference": "reception"
  }
]
```

Key of `oauth2`	Meaning
`url_autorisation`, `url_jeton`	the endpoints of Google; they must be `https`, except on the local loopback
`portee`	the OAuth scope (`https://mail.google.com/`)

Key of `oauth2`	Meaning
`marge_renouvellement_secondes`	the access token is renewed when it has less than this left (`300`)
`delai_secondes`	the delay of each exchange with the authorisation server (`30`)
`delai_consentement_secondes`	how long `--autoriser-google` waits for your answer (`300`)
`chemins_interdits`	folders where a token is refused (`~/Documents`)

5. Consent once, from a terminal on the machine, with the `id` of the source:

```
dibab --autoriser-google imap-google
```

Dibab prints an address that starts with `https://accounts.google.com/`. Open it in a browser. Accept the warning “unverified application”, which is shown because the application is your own, then the consent. The program receives the answer on the local loopback, checks it, exchanges it for a token and writes the token file with mode `600` in a folder with mode `700`. The token file must be outside any git repository and outside `chemins_interdits`; a token file readable by the group or by others stops the source. If the consent is refused, if no answer comes in time, or if Google returns no renewal token, nothing is written; run the command again.

From then on the scan runs as for a password source, without any environment variable: the access token renews itself, and the token appears neither in the journal nor on the screen. A token file that does not exist is not an error: the source is skipped, with the command to run. If Google revokes the authorisation (you removed the access of the application, you changed the password of the Google account, or six months passed without a scan), Dibab stops the source with an explicit message, and you run `--autoriser-google` again.

The scope `https://mail.google.com/` is the only one that Google documents for IMAP, and it allows reading, writing and deleting. That the mailbox stays unmodified is guaranteed by what Dibab sends, which is only the commands listed at the beginning of this chapter, not by the scope. Chapter 10 comes back to it.

Inventory and rotation

Reading what a vault already holds: `--inventaire`

If you already have documents in a vault, filed by hand or by another tool, `--inventaire` reads them and writes a migration plan, without moving, renaming or deleting anything. It is read-only: before and after, each vault is identical. It needs the proof of mount of chapter 5, since each plan is written in its vault.

To have something to read, file the four documents of the trial area again; the chapter 4 rollback put them back in their source:

```
dibab --execute | tail -n 9
```

```
model 1/1, document 1/3
model 1/1, document 2/3
model 1/1, document 3/3
  - notification-organisme.pdf: classe
      -> ~/Classement/Personnel/Administratif/Alice/2026/2026-03-15_Organisme-public_Aides.pdf
  - facture-hebergement.pdf: classe
      -> ~/Classement/Professionnel/Classement/2026/Factures/2026-03-15_Hebergeur-Exemple_Factu⌋
       ↪  res.pdf
  - recette-gateau.pdf: not administrative, stays in place
  - courrier-sans-en-tete.pdf: a_verifier
      -> ~/Classement/Personnel/TO_CHECK/[TO_CHECK]courrier-de-suivi-de-dossier_courrier-sans-e⌋
       ↪  n-tete.pdf

Session report: ~/.local/state/dibab/sessions/rapport_2026-11-23T12:09:03.json
```

Then read the vault of each domain:

```
dibab --inventaire
```

```
Dibab 0.9.0-alpha — session 2026-11-23T12:09:03
  Mode               : inventaire
  Configuration      : ~/.config/dibab/config.json
  Vault personnel     : ~/Classement/Personnel — mounted
  Vault professionnel : ~/Classement/Professionnel — mounted
  Session journal    : ~/.local/state/dibab/sessions/session_2026-11-23T12:09:03.jsonl

Inventory of what exists (read-only, nothing is moved)
  Vault personnel     : 3 listed document(s), 3 in the filed zone, 0 to file, 0 duplicate(s), 0
   ↪  present in the other vault, 1 in TO_CHECK (not listed)
      plan written: ~/Classement/Personnel/logs/inventaire_2026-11-23T12:09:03.json
  Vault professionnel : 1 listed document(s), 1 in the filed zone, 0 to file, 0 duplicate(s), 0
   ↪  present in the other vault, 0 in TO_CHECK (not listed)
      plan written: ~/Classement/Professionnel/logs/inventaire_2026-11-23T12:09:03.json
```

For each vault, the report gives the number of documents read, how many are already in the filed zone, how many are outside it and should be filed, how many are duplicates of another document of the same vault, and how many are also present in the other vault. The plan is a JSON file, `logs/inventaire_<time>.json`, written in the vault it describes:

```
ls ~/Classement/Personnel/logs

inventaire_2026-11-23T12:09:03.json
```

Each vault's plan names only the files of that vault. Of the other vault it keeps a single fact, "this fingerprint is also there", so that the plan of one vault never contains the path of a file of the other.

Every document of the plan receives a proposal:

Proposal	Meaning
nothing to do	the document is in the filed zone
put it in a drop folder, then read the dry-run	the document is outside the filed zone: drop it in a source and let the next pass classify it
identical to another document: to be filed in `duplicate/` after checking	the fingerprint is already present

The folders listed in `inventaire.exclusions` (`logs`, `Archive`, `rebut`, and the folder named by `destination.repertoire_doublon`, in the example), the check folder, the drop folders and the hidden folders are not read.

Keeping the journal short: rotation

Two files grow with use: the action journal, `suivi.journal`, and the folder of the sessions, `suivi.sessions`. The rotation applies the limits of `suivi.rotation` when `--execute` or `--rollback` starts:

Key	Effect
`journal_taille_max_mo`	when the journal exceeds this size, it is renamed `journal.1.jsonl`, the older archives move up by one number and a new journal starts
`journal_archives_max`	the number of archives kept; the oldest one is removed when a new one would exceed it
`sessions_max`	the number of session journals and reports kept; the oldest ones are removed

This is the only deletion Dibab performs, and it concerns its own journals, never a document. An archive that still holds an action that has not been undone is never removed: the rotation waits rather than lose what `--rollback` must be able to read. Undo a filing you have doubts about before its journal grows old.

The files of the state are written with mode `600` and their folder with mode `700`. Where they are kept is the choice described in chapter 5.

Diagnosis

`--diagnose` checks the installation, the configuration, Ollama and the vault of each domain, one line per check, and writes nothing: no file is created, nothing is moved, no lock is taken. It opens a single connection, to Ollama, to read the list of models. It opens none to a remote machine or to a mailbox, so it is safe to run at any time. It plays every check to the end, whatever the result of the previous ones.

```
dibab --diagnose

OK python : 3.13.5
OK outil_pdftotext : /usr/bin/pdftotext
OK outil_pdftoppm : /usr/bin/pdftoppm
OK outil_findmnt : /usr/bin/findmnt
OK outil_ssh : /usr/bin/ssh
OK configuration : ~/.config/dibab/config.json
AVERTISSEMENT emplacement_etat : the state and the journal are in clear text, outside any vault:
 ↪  ~/.local/state/dibab
OK ollama : http://127.0.0.1:18434 answers, 3 model(s) installed
OK modele_ocr : essai-ocr:latest
OK modele_tri_rapide : essai-tri:latest
OK modele_classification : essai-classification:latest, 1 model(s) in the consensus
NON_CONFIGURÉ modele_tri_decision : ollama.roles.tri_decision absent or empty: the triage goes
 ↪  through ollama.roles.tri_rapide
OK destination_personnel : ~/Classement/Personnel
OK destination_professionnel : ~/Classement/Professionnel
Diagnostic: 12 OK, 1 warning(s), 0 failure(s), 1 not configured, 0 not checked
```

Each line reads `<STATE> <identifier> : <message>`, then a summary closes the report. The message is in the language of your session; the states and the identifiers never change with the language, so that a script can compare them.

The states

State	Meaning
`OK`	the check passed
`AVERTISSEMENT`	a defect that prevents no mode, or a defect of a component that this instance does not use
`ÉCHEC`	the check failed on something that the installation or the configuration found requires
`NON_CONFIGURÉ`	the configuration that the check reads does not exist yet
`NON_VÉRIFIÉ`	the check depends on another one that did not pass, or on a missing configuration

The exit code is `0` when there is no `ÉCHEC` and `1` when there is at least one. A warning, a missing configuration and an unchecked item do not change it.

The checks, in order

Identifier	What is checked	If it is not OK
python	the interpreter is 3.10 or later	ÉCHEC below
outil_pdftotext, outil_pdftoppm, outil_findmnt	the tool is executable in the PATH	ÉCHEC if absent
outil_ssh	ssh is executable in the PATH	ÉCHEC if the configuration declares sources_distantes, AVERTISSEMENT otherwise
configuration	a configuration is found (chapter 2) and valid: readable JSON, the keys that each mode requires	none found: NON_CONFIGURÉ, and the message lists the four places tried; found but invalid: ÉCHEC, with the missing keys
emplacement_etat	the state and the journal are under the root of a declared vault	outside any vault: AVERTISSEMENT (the state is in clear text)
ollama	GET <ollama.url>/api/tags answers within ollama.timeout_secondes	no answer: ÉCHEC
modele_ocr, modele_tri_rapide, modele_classification	the role has at least one installed model among its preferences	none installed or an empty list: ÉCHEC, and the message names ollama.roles.<role>
modele_tri_decision	the optional decision triage (chapter 3): first installed model of ollama.roles.tri_decision, decision capability, valid thresholds in classification.seuils_tri_decision, an installed model for ollama.roles.objet	role absent or empty: NON_CONFIGURÉ; declared but unusable: AVERTISSEMENT, the message names the key to correct, and the triage goes through tri_rapide
destination_personnel, destination_professionnel	the proof of mount of chapter 5: the file system type of the root is accepted and the root is writable	not proven: ÉCHEC, and the message gives the reason

A source with a password whose environment variable is not defined is reported with a warning in the check configuration, since the program skips such a source at run time. No check writes a secret, a document name or the content of an environment variable.

Reading a failure

A configuration of the trial area declares a vault for each domain that must be `fuse.gocryptfs`, while the directories are ordinary ones, which is what a closed vault looks like:

```
dibab --diagnose --config ~/.config/dibab/config-ferme.json; echo "exit code: $?"

OK python : 3.13.5
OK outil_pdftotext : /usr/bin/pdftotext
OK outil_pdftoppm : /usr/bin/pdftoppm
OK outil_findmnt : /usr/bin/findmnt
OK outil_ssh : /usr/bin/ssh
OK configuration : ~/.config/dibab/config-ferme.json
AVERTISSEMENT emplacement_etat : the state and the journal are in clear text, outside any vault:
 ↪  ~/.local/state/dibab
OK ollama : http://127.0.0.1:18434 answers, 3 model(s) installed
OK modele_ocr : essai-ocr:latest
OK modele_tri_rapide : essai-tri:latest
OK modele_classification : essai-classification:latest, 1 model(s) in the consensus
NON_CONFIGURÉ modele_tri_decision : ollama.roles.tri_decision absent or empty: the triage goes
 ↪  through ollama.roles.tri_rapide
ÉCHEC destination_personnel : vault not mounted (type ext4, expected fuse.gocryptfs) (~/Classem
 ↪  ent/Personnel)
ÉCHEC destination_professionnel : vault not mounted (type ext4, expected fuse.gocryptfs) (~/Cla
 ↪  ssement/Professionnel)
Diagnostic: 10 OK, 1 warning(s), 2 failure(s), 1 not configured, 0 not checked
exit code: 1
```

The two checks `destination_*` are in failure and the code is `1`, while Ollama, the models and the tools are fine. The message tells what to do: open the vault, or write the right type in `systemes_fichiers_acceptes`.

Other frequent failures:

Line	What to do
`ÉCHEC ollama`	the service does not answer: check `ollama.url`, then that Ollama runs (`systemctl status ollama`)
`ÉCHEC modele_<role>`	correct `ollama.roles.<role>` according to `ollama list`
`AVERTISSEMENT modele_tri_decision`	install the decision model or the object model named, or write its thresholds; until then the triage goes through `tri_rapide`
`ÉCHEC destination_<domain>`	open the vault named, or correct the accepted types
`ÉCHEC configuration`	add the keys named, copying the block from `config.exemple.json`

A fresh installation

Right after the installation, with no configuration and no Ollama, the diagnosis is not a failure: a missing configuration is the normal state of a fresh installation and is named `NON_CONFIGURÉ`, not confused with a breakdown. In an empty home directory:

```
HOME="$(mktemp -d)" dibab --diagnose; echo "exit code: $?"

OK python : 3.13.5
OK outil_pdftotext : /usr/bin/pdftotext
OK outil_pdftoppm : /usr/bin/pdftoppm
OK outil_findmnt : /usr/bin/findmnt
OK outil_ssh : /usr/bin/ssh
NON_CONFIGURÉ configuration : no configuration found (the --config option: not given; the DIBAB
↪  _CONFIG variable: not given; the XDG location: ~/.config/dibab/config.json; the script
↪  folder: ~/applications/dibab/config.json)
NON_VÉRIFIÉ emplacement_etat : depends on the configuration
NON_VÉRIFIÉ ollama : depends on the configuration
NON_VÉRIFIÉ modele_ocr : depends on the configuration
NON_VÉRIFIÉ modele_tri_rapide : depends on the configuration
NON_VÉRIFIÉ modele_classification : depends on the configuration
NON_VÉRIFIÉ modele_tri_decision : depends on the configuration
NON_VÉRIFIÉ destination_personnel : depends on the configuration
NON_VÉRIFIÉ destination_professionnel : depends on the configuration
Diagnostic: 5 OK, 0 warning(s), 0 failure(s), 1 not configured, 8 not checked
exit code: 0
```

The tools are `OK`, the configuration is `NON_CONFIGURÉ` and the eight checks that depend on it are `NON_VÉRIFIÉ`. The exit code is `0`. Once a configuration exists, everything it declares and the machine does not hold is an `ÉCHEC`: a diagnosis that returned `0` with Ollama stopped would hide the very breakdown that you came to find.

In another language

The language follows `LC_ALL`, then `LC_MESSAGES`, then `LANG`:

```
LC_ALL=fr_FR.UTF-8 dibab --diagnose | tail -n 1

Diagnostic : 12 OK, 1 avertissement(s), 0 échec(s), 1 non configuré(s), 0 non vérifié(s)
```

Sovereignty and limits

What the tool guarantees

- **No document leaves your network**. The calls to the language models are plain HTTP requests to `ollama.url` (by default `http://localhost:11434`). No document and no file name goes anywhere else, and there is no telemetry.
- **The only outside services the code can reach are the ones you declare**: your machines over `ssh`, your IMAP mailboxes and, if you enable OAuth2, the authorisation endpoints of Google.
- **No secret is stored in `config.json`**. A mail password lives in an environment variable, a Google token in a file with mode `600` outside any repository. A token is never written in a journal, shown on the screen or put in an error message.
- **A document is never lost by a filing**. It is copied, the copy is checked by SHA256, and only then is the original removed. Every filing is written in the action journal and can be undone with `--rollback`.
- **A mailbox and a remote machine are left untouched**. The IMAP commands that are sent are the ones listed in chapter 7; a remote machine is only read, and nothing is removed there.
- **The text of a document is data, never an instruction**. It is delimited by a random marker and the model answers a closed JSON that Dibab validates. An answer that does not fit the schema counts as "uncertain" and leads to the check folder. A model can never trigger an action.
- **Doubt leads to a human decision**. An uncertain domain, models that disagree, or a common piece without a unanimous category put the document in `TO_CHECK/`, not in a risky place.
- **The journal holds paths and categories, never text**. It does hold file names and senders, which already reveal a lot; keep it in a vault (chapter 5).

The code is released under the GNU Affero General Public License, version 3. It uses only the Python standard library, so there is no third-party package to audit.

Respect of the household

A machine, a directory or a mail account enters the scope of the tool only with the agreement, explicit or implicit, of the person it concerns. The software cannot check this: the administrator of the instance commits to it. Do not point Dibab at the mailbox or the files of a person who does not know that they are read.

The limits

- **Remote sources have been tried only against fake servers**. The `ssh` source and the IMAP source, with a password or with OAuth2, are covered by tests and by the replays of this manual, which use a fake `ssh`, a fake IMAP server and a fake authorisation server. Their use on a real machine of the household and a real mailbox, including the consent with Google, remains to be done: until that trial is made, treat these sources as not proven in real conditions and begin with `--dry-run`.
- **The quality of the filing is the quality of your models**. On the reference instance, a first trial on 15 real documents filed 2, set 3 aside as not administrative and put 10 in `TO_CHECK/`, out of caution. Measure before trusting; the measurement tool of the sources, `_outils/mesurer_domaines.py`, samples filed documents and compares the verdict of the models with the vault in which each one sits.
- **Linux only**. The proof of mount relies on `findmnt`, which belongs to `util-linux`.

- **Dibab does not mount, encrypt or synchronise anything**, and it does not manage the disk space or the backups: this is up to you.
- **The Google scope is wider than the use**. The scope `https://mail.google.com/` is the only one that Google documents for IMAP. It allows reading, writing and deleting. That the mailbox stays unmodified rests on the commands that Dibab sends, which are listed in chapter 7, not on the scope.
- **The defaults are examples**. The categories, the known senders and the naming template of `config.exemple.json` are generic; adapt them to your country.
- **The steps in the Google console depend on Google's interface**, which changes. The notions remain those of its OAuth2 documentation.
- **The project is young**. The version is 0.9.0-alpha; there is no continuous integration. The tests of the sources are run by hand.
- **Ollama is not part of Dibab**. It is never installed and never updated by it; a model that is renamed or removed upstream is a line to correct in `ollama.roles`.

Reporting a problem

A vulnerability is reported in private, according to the file `SECURITY.md` of the sources, never in a public issue. For a contribution, read `CONTRIBUTING.md` in the same place.