

Petacore

A workshop for Linux desktop software.

Write it, test it, sign it, ship it.

Project documentation

kotechsoft.com · hello@kotechsoft.com

github.com/petacomm/petacore

GNU GPL v3 · Petacomm I.S.

Contents

- 1 What Petacore is**
 - 1.1 The problem
 - 1.2 The approach
 - 1.3 Who it is for
- 2 What it does**
 - 2.1 The workspace
 - 2.2 Writing code
 - 2.3 Keeping history
 - 2.4 Trying things safely
 - 2.5 Producing packages
 - 2.6 Planning work
 - 2.7 Understanding a project
- 3 How it works**
 - 3.1 Design principles
 - 3.2 Architecture
 - 3.3 Where things are stored
 - 3.4 Dependencies
- 4 Security**
 - 4.1 Credentials
 - 4.2 Signing keys
 - 4.3 Isolation
 - 4.4 Cloud storage
 - 4.5 Testing
- 5 Installation**
 - 5.1 Requirements
 - 5.2 Installing
 - 5.3 Removing
- 6 Distribution**
 - 6.1 Packages
 - 6.2 Metadata
 - 6.3 Signatures
 - 6.4 Repositories
- 7 Scope of this release**
 - 7.1 In this release
 - 7.2 Not in this release
 - 7.3 Known limitations
- 8 Contributing**
 - 8.1 Reporting problems
 - 8.2 Working on the code
 - 8.3 Translations
- 9 Licence and contact**

What Petacore is

Petacore is a desktop application for people who write software for Linux. It brings the ordinary work of a project into one window: editing files, keeping a history, trying changes without risk, building a package, signing it, and publishing it.

1.1 The problem

Shipping a small Linux application takes more tools than writing one. A developer working alone assembles a version control client, an editor, a terminal, a packaging script, a signing key, and somewhere to put the result. Each of these is simple on its own. Together they are a day of reading manual pages, and the knowledge does not transfer: the next project starts from the same place.

The cost falls hardest on the last step. Writing the code is the part people enjoy and understand. Producing a package that installs cleanly, carries a signature, and tells a software centre who made it is the part that gets postponed, and then skipped.

1.2 The approach

Petacore does not replace those tools. It drives them. Underneath are git, dpkg, rpmbuild, GnuPG, bubblewrap and rclone, each doing the job it already does well. What Petacore adds is a single place to reach them, with sensible defaults and the sharp edges guarded.

The design follows from one idea: a developer should be able to go from an empty folder to a signed package without leaving the application, and without having read anything first.

1.3 Who it is for

Petacore is built for developers working alone or in small teams, on projects they own. It assumes familiarity with a file system and with the shape of a software project. It does not assume fluency at the command line, and nothing in the interface requires it.

Petacore is not a team platform. There is no shared server, no access control and no review workflow. Projects belong to the person at the keyboard.

What it does

The application is arranged as a set of pages, each covering one stage of the work. This section describes what each one is for.

2.1 The workspace

A project in Petacore is an ordinary folder with a git repository in it. Nothing is hidden in a database, and nothing about the folder is particular to Petacore: another tool can open it, and removing Petacore leaves the project intact.

The Project page is a file manager for that folder. It searches, filters by kind, moves files by drag and drop, and undoes what it did. Deletions can be reversed. Holding a modifier key over a file shows a preview without opening it.

2.2 Writing code

The Editor opens files in tabs with syntax highlighting. It is a small editor rather than a general one: enough to change a file, read an error and move on, without becoming a place to live.

Two features exist because long files are hard to hold in the head. Sections let a range of lines be named; selecting one dims everything outside it. Saving offers to copy the file into the folder of the installed application, so a change can be tried immediately in the running program.

2.3 Keeping history

Snapshots are compressed copies of the project folder, taken on request and on a timer. They sit alongside git rather than replacing it: git records what was committed, snapshots record what was on disk, including the work that was not ready to commit.

Restoring takes a snapshot of the current state first, so a restore can itself be undone.

2.4 Trying things safely

The Sandbox runs a shell in an isolated environment built with bubblewrap. The root file system is mounted read only, the home directory is masked, and the process, IPC and hostname namespaces are separated. A project can be copied in, run, and left in whatever state it ends in; closing the session removes everything.

The network is closed by default. When something inside tries to reach it and fails, Petacore notices and asks whether to open it for that session.

2.5 Producing packages

The Package page turns a project into an installable file. Debian packages are built with dpkg-deb and RPM packages with rpmbuild. Both carry AppStream metadata, so a software centre can show who published the package and under what licence.

Signing is on by default. The signature is placed inside the package itself, so a single file carries its own proof of origin.

2.6 Planning work

Next Updates is a list of what is still to do, kept inside the project rather than in a separate service. Each entry has a title, a note and a priority, and can carry an attached document. The list travels with the project and syncs with it.

2.7 Understanding a project

The Overview page reports what a project is made of: how many lines and characters of code, how many files, which languages by share of the code, and which operating system it appears to target. The same figures can be rendered as an image to post.

Page	Purpose
Overview	Statistics and the shareable summary
Project	Files: search, move, undo
Editor	Tabbed editing with sections and focus
Snapshots	Timestamped copies of the folder
Terminal	A shell in the project directory
Sandbox	An isolated environment for trying things
Package	Building and signing packages
Keys	Managing signing keys
Next Updates	Plans, with attachments

How it works

3.1 Design principles

Four rules decided most of the arguments during development.

- Drive standard tools. Where a well understood program exists, Petacore calls it rather than reimplementing it. The output is ordinary and can be inspected with anything else.
- Leave the project alone. Everything Petacore adds lives in a single hidden folder inside the project. Delete it and the project is unchanged.
- Say what is happening. Operations that take time report progress. Operations that cannot be undone ask first. Operations that were refused explain why.
- Fail visibly, not silently. A missing dependency disables the feature that needs it and says so, rather than appearing to work.

3.2 Architecture

The application is written in Python 3 with GTK 4 and libadwaita. The interface is separated from the work it drives: the modules that talk to git, GnuPG, dpkg and rclone have no knowledge of the interface, and can be exercised on their own. Long operations run on worker threads so the window stays responsive.

Module	Responsibility
window, pages, dialogs	The interface
gitops, github	Version control and the GitHub connection
gdrive	Cloud synchronisation, through rclone
debuild	Package construction and metadata
gpgsign	Key management and signatures
sandbox	Isolated sessions
snapshots	Timestamped copies
crypto	Encryption of the cloud copy
stats, sharecard	Project statistics and the summary image
secrets	Credentials, held in the desktop keyring

3.3 Where things are stored

Location	Contents
~/.config/petacore/	Settings and the project list
~/.local/share/petacore/	Snapshots
<project>/petacore/	Plans, attachments, editor sections
Desktop keyring	The GitHub credential
~/.gnupg/	Signing keys, managed by the system

The project folder itself holds nothing else. A project can be moved, copied or archived with ordinary tools.

3.4 Dependencies

Petacore depends on programs that are packaged in Ubuntu and Debian. Each is optional in the sense that its absence disables one feature rather than the application.

Program	Used for
git	Version control
dpkg, rpm	Building packages
gnupg	Signing
rclone	Google Drive synchronisation
bubblewrap	Sandbox isolation
librsvg2-bin	Rendering the summary image
gh	Reusing an existing GitHub session

Security

Petacore holds a credential that can push to a repository, a key that can sign software in the user's name, and a copy of the project in cloud storage. This section states plainly how each is handled.

4.1 Credentials

The GitHub access token is stored in the desktop keyring through libsecret, not in a configuration file. Where no secret service is available it falls back to a file readable only by the owner, and the interface says which of the two is in use.

4.2 Signing keys

The private half of a signing key never leaves the machine. Cloud synchronisation backs up public keys only, which is all another computer needs to verify a package.

If the key selected for signing is no longer present, Petacore stops rather than substituting another one. Signing with a key the user did not choose would put a different identity on the package with nothing to draw their attention to it.

The full fingerprint of the signing key is shown wherever the key is named. A name and an address can be shared by several keys; a fingerprint cannot.

4.3 Isolation

The Sandbox starts from an empty environment. Inheriting the caller's environment would carry the SSH and GPG agent sockets into the jail, and with them the ability to authenticate and sign as the user. Only the display variables are put back.

The isolation is designed to prevent accidental damage from code under development. It is not a defence against software written to escape it. For that, use a virtual machine.

4.4 Cloud storage

Synchronisation makes the copy in Drive match the local folder. Files removed by that operation are moved into a dated folder rather than deleted, so a mistake can be reversed from the cloud copy itself.

Before an upload that is smaller than what is already stored, Petacore asks for confirmation. That case usually means another computer holds work this one has not fetched.

A project can be encrypted for the cloud. Encryption is applied on the way up and undone on the way down: file contents and file names are both unreadable in Drive, while the copy on disk stays ordinary

source code.

4.5 Testing

Security behaviour is covered by a regression suite that ships with the source. It checks that credentials stay out of configuration files, that package metadata cannot be injected, that credential files are excluded from packages, that the sandbox environment is cleared, and that tampering with a signed package invalidates its signature.

```
python3 security_tests.py
```

5

Installation

5.1 Requirements

Item	Requirement
Operating system	Ubuntu 24.04 or newer, and Debian derivatives
Architecture	x86-64
Python	3.10 or newer
Desktop	GNOME
Memory	2 GB. Above 8 GB, plan attachments are served from memory

5.2 Installing

Clone the repository and run the installer as yourself. It asks for administrator rights only for the packages it needs, and refuses to run as root: everything it installs belongs in your own account.

```
git clone https://github.com/petacomm/petacore.git
cd petacore
./install.sh
```

5.3 Removing

The uninstaller asks separately about the application, the snapshots and the settings. Snapshots are the only copies of earlier states of a project, so removing the program does not remove them unless asked.

```
./uninstall.sh
```

Distribution

6.1 Packages

Petacore builds Debian and RPM packages from a project folder. Files that carry credentials are excluded, and the omission is reported rather than done silently.

6.2 Metadata

Every package carries AppStream metadata: the publisher, an SPDX licence identifier, a summary and a home page. Without it a software centre shows a package as coming from an unknown publisher under an unknown licence, whatever the Debian control file says.

6.3 Signatures

Signatures are embedded in the package. A Debian package is an archive; the signature covers its members and is stored as a further member, so one file carries both the software and the proof of who built it.

6.4 Repositories

A single package handed to someone is always treated as untrusted, and there is no way to deliver an update afterwards. A repository solves both. Petacore can assemble one: a pool of packages, signed indexes, and the public key users need in order to trust them. The result is a directory of static files that can be served from anywhere.

```
# What a user runs once, to trust and add the archive
curl -fsSL https://example.com/apt/keyring.asc \
| sudo gpg --dearmor -o /usr/share/keyrings/vendor.gpg
echo "deb [signed-by=/usr/share/keyrings/vendor.gpg] \
https://example.com/apt stable main" \
| sudo tee /etc/apt/sources.list.d/vendor.list
```

Scope of this release

This section describes what this release contains, and what it deliberately does not.

7.1 In this release

- A GNOME desktop application, built with GTK 4 and libadwaita.
- Project management, file browsing, editing, snapshots and a terminal.
- An isolated environment for running work in progress.
- Debian and RPM package construction with AppStream metadata and embedded signatures.
- GPG key generation and management.
- Synchronisation with GitHub and with Google Drive, with optional encryption of the cloud copy.
- Five interface languages.

7.2 Not in this release

- Other desktop environments. This release targets GNOME. Supporting a second environment properly means matching its design conventions rather than approximating them, and that work has not been done.
- Distributions beyond Ubuntu and Debian. The installer expects apt. Other package managers are not supported.
- Team features. There is no shared server, no access control and no review workflow.
- Automatic updates. A new release is installed the same way as the first.

7.3 Known limitations

- The Sandbox notices an attempt to reach the network by reading the error a blocked program prints. Software that fails silently is not detected.
- The target platform shown on the Overview page is inferred from file names and file contents. It is an estimate.
- A package installed from a file, rather than from a repository, is marked as coming from a third party by software centres. Only a repository removes that warning.

Contributing

8.1 Reporting problems

Problems and suggestions belong in the issue tracker at github.com/petacomm/petacore. A useful report says what was expected, what happened instead, and enough detail to reproduce it. Starting the application from a terminal and including the messages it prints usually settles the question quickly.

8.2 Working on the code

The source is Python with no build step. Run it from a clone, and the security suite before sending a change.

```
git clone https://github.com/petacomm/petacore.git
cd petacore
python3 -m petacore.main # run from the source tree
python3 security_tests.py # before opening a pull request
```

Changes that touch packaging, credentials, the sandbox or synchronisation should come with a test in that suite.

8.3 Translations

All interface text lives in one module, as a table of keys against languages. Adding a language means adding a column; adding a string means adding a row in every language. The application refuses to start with an incomplete table, which keeps translations from silently falling behind.

Licence and contact

Project	Petacore
Application ID	io.petacore.Petacore
Licence	GNU General Public Licence, version 3 (GPL-3.0-only)
Source	https://github.com/petacomm/petacore
Website	https://kotechsoft.com
Contact	hello@kotechsoft.com

Licence

Petacore is free software, published under the GNU General Public Licence, version 3. You may use it, study it, change it and pass it on. If you distribute a modified version, it must carry the same licence and its source must be available.

The full text is in the repository, in the file named COPYING, and at gnu.org/licenses/gpl-3.0.html.

The licence covers Petacore itself, not what you build with it. Packages produced by Petacore carry whatever licence their author chooses; using a GPL-licensed tool does not place that licence on your work.

Petacore – a workshop for Linux desktop software
Copyright (C) 2026 Petacomm I.S.

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

A note on this document

This document describes the software as built. Where a limitation exists it is stated rather than omitted, because a developer choosing a tool is better served by an accurate account than a flattering one.

Petacore is developed by Petacomm I.S.

kotechsoft.com · hello@kotechsoft.com · github.com/petacomm/petacore
Copyright (C) 2026 Petacomm I.S. Released under the GNU GPL, version 3.