

User manual

Find what is wrong with a music library, and fix it reversibly.

VERSION 0.21.0

phorel

© 2026 Phorel
www.phorel.net

phorel

luthier 0.21.0

User manual

Copyright © 2026 Phorel. All rights reserved where not granted below.
https://www.phorel.net · mail@phorel.net

luthier is freeware: free to use, and to pass on unmodified and free of charge. It comes WITHOUT ANY WARRANTY. See Appendix A for the licence and Appendix B for the third-party software luthier is built on.

This manual describes luthier 0.21.0 on Linux. Example paths such as `/music` stand for your own library. Numbers quoted from “one real library” come from the 38,000-track collection luthier was built against.

Lyrion, Navidrome, FLAC, iTunes, Deezer and other names mentioned are the property of their respective owners and are used only to identify them.

Set in Nimbus Sans and JetBrains Mono.

Contents

1 **Introduction** 5

1.1 What luthier does 5
1.2 How it keeps your library safe 5
1.3 What luthier is not 6
1.4 Conventions in this manual 6

2 **Getting started** 7

2.1 Requirements and installation 7
2.2 First run 7
2.3 Your first clean-up 7

3 **The console** 9

3.1 Starting the console 9
3.2 The screen 9
3.3 Keyboard reference 10
3.4 The Library tab 10
3.5 The Snapshots tab 11
3.6 The Fixes tab 12
3.7 Reviewing: findings and preview 14
3.8 Applying 16
3.9 Stale baselines 16
3.10 Undoing changes 17
3.11 Other findings 17
3.12 The Import tab 19
3.13 The Settings tab 20

4 **The fixes in detail** 21

4.1 Stray bytes · clean 21
4.2 Non-standard tag keys · aliases 21
4.3 Artist spellings · artists 22
4.4 Albums with no tags · filenames 22
4.5 Missing artists · fill 23
4.6 Missing album artists · albumartist 23
4.7 Empty fields from MusicBrainz · mbfill 24
4.8 Check all against MusicBrainz · mbcheck 24
4.9 Genre sprawl · genres 24
4.10 Versions of the same album · editions 25
4.11 Folder names · folders 25

5 **Finding problems without fixing them** 27

5.1 What state is the library in? · report 27
5.2 Finding damage · verify 27
5.3 Finding duplicates · duplicates 27
5.4 Do I already have this? · arrivals 28
5.5 Comparing a library with its mirror · compare 28
5.6 Album art · artwork 28

6 **Command-line reference** 29

6.1 Conventions 29
6.2 Commands at a glance 29
6.3 Snapshots and restoring 30
6.4 Inspecting 31
6.5 The passes that write 32
6.6 Importing new music from the command line 34

7 **Reference** 35

7.1 Supported formats 35
7.2 How restores find files 35
7.3 The snapshot file 35
7.4 Folders the passes leave alone 36
7.5 Files and locations 36
7.6 Performance 36
7.7 Troubleshooting 37

A **Licence** 38

B **Third-party software and credits** 39

Index 42

CHAPTER 1

Introduction

A luthier repairs instruments. You bring in something damaged; a careful luthier hands it back working, and nothing else about it changed. That is the job this program does for a music library.

1.1 What luthier does

A large music collection goes wrong slowly. One artist ends up spelled three different ways. Five genre strings mean the same thing. A compilation shatters into forty albums because nobody ever wrote an album artist. An SACD rip and a vinyl rip of the same record show up as two identical rows. None of it is visible until you browse the collection, and none of it is safe to fix by hand at scale, because bulk tag edits are exactly the kind of change you cannot easily take back.

luthier looks at the tags of every file in the library, tells you what is wrong, shows you every row behind each number, and writes only what you tick. It works in two ways:

- **The console**: a full-screen terminal interface for the part that is hard to get right from a command line, which is deciding what to run. It is what opens when you type `luthier`.
- **The command line**: every job the console does is also a subcommand, for scripts, scheduled jobs and machines without a full terminal.

1.2 How it keeps your library safe

Three rules run through everything luthier does.

1. **A snapshot comes first.** Before anything is written, luthier records every tag in the library to a small file. Any fix applied afterwards can be undone from it.
2. **Nothing is written unseen.** Every fix is a dry run until you apply it. You can read every change it would make, in the order it would make them, before a single file is touched.
3. **Judgement stays with you.** Where the evidence is ambiguous, luthier reports instead of guessing. Which duplicate to delete, which cover to use and which spelling a band chose for itself are your decisions.

Why a tag snapshot rather than a filesystem snapshot

Filesystem snapshots are the usual way to make a change reversible, but they need btrfs or ZFS and cost as much disk as the library itself. Most large collections live on an external NTFS or ext4 drive with nothing to spare. luthier records only the metadata: a snapshot of a 38,000-track library is a few tens of megabytes and takes minutes to make.

It also survives retagging. FLAC stores an MD5 checksum of its *decoded* audio, which does not change when you edit tags. A restore matches files on that checksum, so one snapshot stays valid across as many tag edits as you like, and refuses to overwrite a file whose audio itself has changed.

1.3 What luthier is not

luthier is a library *fixer*, not a library manager. It does not browse or play music; servers such as Navidrome and Lyrion do that far better. It changes tags, and the only thing it ever moves is album folders, through the Folder names fix, which logs every rename so it can be undone.

> **NOTE**
> luthier is one part of a larger project: a bit-perfect, multi-room music setup built entirely from free and open-source software. The guide to the whole setup is on www.phorel.net.

1.4 Conventions in this manual

LOOKS LIKE	MEANS
a	A key to press in the console.
Preview changes	A button, tab or field on screen, named as it appears.
`luthier report`	Something to type in a terminal, or a file name.
`ALBUMARTIST`	A tag field, written in capitals as most taggers show it.
SAFE REVIEW FIRST CAREFUL	How far to trust a fix. See Risk levels.

CHAPTER 2

Getting started

Install luthier, point it at your library once, and work through a first clean-up: snapshot, analyse, choose, preview, apply.

2.1 Requirements and installation

The single-file executable

luthier for Linux ships as one executable that contains everything it needs, Python included. Put it somewhere on your `PATH` and make it executable:

```
install -m 755 luthier ~/.local/bin/luthier
luthier --version
```

The executable unpacks itself once per version into `~/.cache/luthier/version`, so the very first start takes a few seconds and later ones do not.

Optional: flac and ffmpeg

Two external programs are used, if they are installed, to check that files actually decode (verify, and **Check if it decodes** in the console). Everything else works without them. On most distributions:

```
sudo pacman -S flac ffmpeg        # Arch, CachyOS, Manjaro
sudo apt install flac ffmpeg      # Debian, Ubuntu
```

2.2 First run

Type `luthier`. A splash screen shows the version for a moment; any key skips it. The first time, with nothing saved, the console opens on the **Settings** tab and asks for two folders:

- **Library location**: the top folder of your music, for example `/music`.
- **Snapshot location** (labelled **Tagsnap location**): where snapshots are kept. Put it on a different drive from the library if you can.

Press **Save**. The settings take effect the next time the console opens, so quit with `q` and start it again; from then on `luthier` opens straight into your library.

2.3 Your first clean-up

1. **Take a baseline.** On the **Snapshots** tab press **New snapshot**, or `s` anywhere. Every tag in the library is recorded before anything is written. On a spinning disk this takes around ten minutes for 40,000 tracks.

2. **Analyse.** Press `a`. The counts on the **Fixes** tab fill in pass by pass, and the **Library** tab sums up what was found.
3. **Choose what to fix.** Tick fixes on the Fixes tab. Read each one's **Impact** section, untick anything it gets wrong, and press `d` or **See all findings** to see every row behind the count.
4. **Preview.** Press `p` to list every write the selection would make, in the order it would make them. Nothing you unticked appears.
5. **Apply.** When it reads right, press `r` or **Apply selected**, and confirm.

Changed your mind? **Restore from this** on the Snapshots tab writes out the exact command that puts the tags back (see Undoing changes).

The same session from the command line, for comparison:

```
# 1. the safety net, before anything else
luthier snapshot /music -o ~/tags-2026-10-15.jsonl.gz

# 2. what state is this library actually in?
luthier report ~/tags-2026-10-15.jsonl.gz

# 3. run a pass as a dry run, read it, then apply it
luthier albumartist ~/tags-2026-10-15.jsonl.gz --root /music
luthier albumartist ~/tags-2026-10-15.jsonl.gz --root /music --apply

# 4. changed your mind
luthier restore /music ~/tags-2026-10-15.jsonl.gz --only ALBUMARTIST --apply
```

CHAPTER 3

The console

The console shows what is wrong, what each fix would change and how far to trust it, and only then offers to write.

3.1 Starting the console

```
luthier                                          # the library saved in Settings
luthier ui /music --snapshots ~/music-tag-backups  # another library, this session only
luthier ui --snapshot ~/tags-2026-10-15.jsonl.gz   # plan from a particular snapshot
```

Run without arguments, the console opens the library saved on the Settings tab and plans from the newest snapshot of it. `luthier ui root` overrides the saved library for one session without changing what is saved.

3.2 The screen

Five tabs run along the top: **Library**, **Snapshots**, **Fixes**, **Import** and **Settings**. Click a tab, or move to the tab bar and use the arrow keys. Under every tab sit three lines that never move:

- **The session strip** says which snapshot this session is planning from, how old it is, the scope, and the library. Its second line names the next step, for example “press p to read every change the 3 selected would make”, or warns that the baseline is stale (see Stale baselines).
- **The activity line** shows the last thing that happened. Press `l` to open the whole log; it opens by itself when something fails.
- **The key bar** lists the keys that work everywhere.

While a long job runs, a light sweeps along the status line beside the progress bar. The same indicator is used for every long job, so a pass that reports progress in bursts still reads as working rather than stuck.

3.3 Keyboard reference

Anywhere in the console

`a`	Analyse the library
`d`	See all findings for the highlighted fix
`p`	Preview changes
`r`	Apply selected
`s`	New snapshot
`l`	Open or close the activity log
`Delete`	Delete the selected snapshot (Snapshots tab)
`Ctrl`+`p`	Command palette
`q`	Quit

In a findings table

`Space`	Include or leave out the highlighted row
`e`	Open the table in the browser
`Esc`	Close

In the artwork window

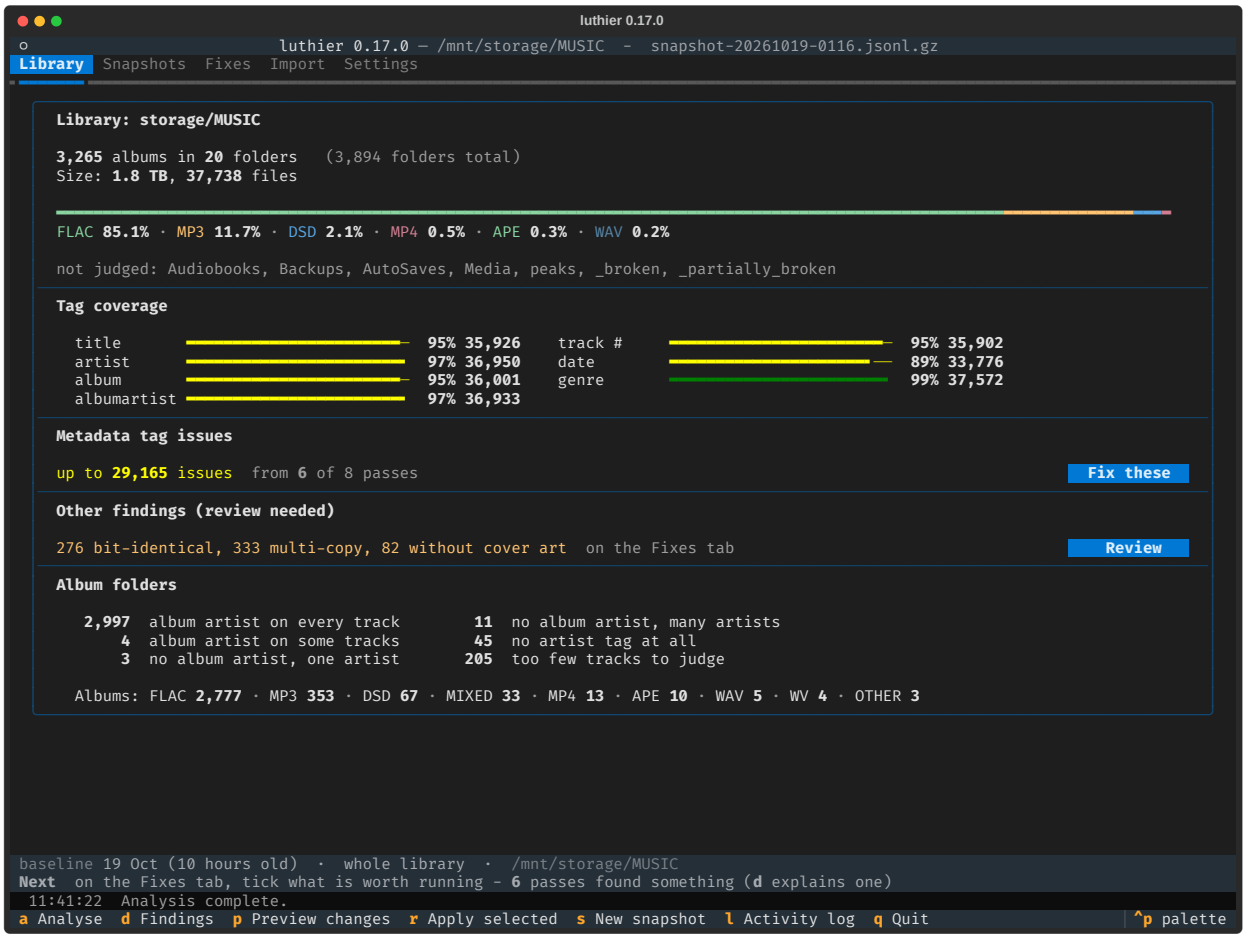
`a` / `n`	Select all / select none
`Esc`	Close

In any other window

`Esc`	Close or cancel. The genre editor closes without saving.

`Tab` and `Shift`+`Tab` move between controls, `Enter` presses the focused button, and `Space` ticks the focused checkbox. Everything can also be clicked with the mouse.

3.4 The Library tab



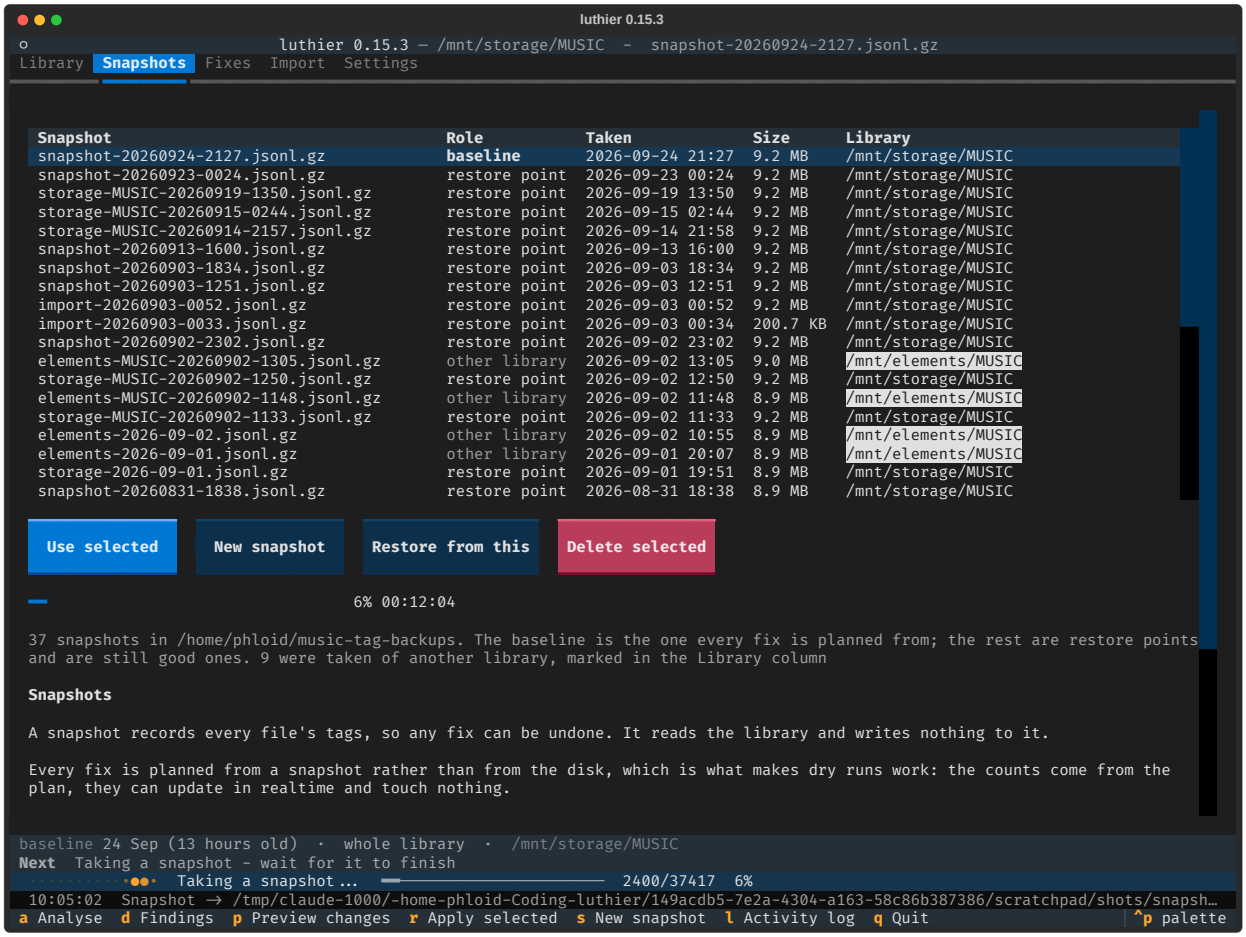
The Library tab. What the library holds, tag coverage per field, what the passes found, the findings that need a person, and album folders by what they are missing.

One page in five sections:

- **The library**: albums, folders and size, and what it is made of as a bar with one colour per format.
- **Tag coverage**: for each important field, how many files carry it.
- **Metadata tag issues**: a one-line count of what the passes found, with **Fix these** to jump to the Fixes tab.
- **Other findings (review needed)**: a one-line count of the problems no pass will fix for you, with **Review** to jump to them on the Fixes tab. See Other findings.
- **Album folders**: folders by what they are missing.

The reference sections pair up into two columns when the terminal is wide enough, so the whole page fits on one screen. The folders every pass ignores are named here too, so a count never quietly leaves something out (see Folders the passes leave alone).

3.5 The Snapshots tab



The Snapshots tab while a new snapshot is being taken.

Lists the snapshots in the snapshot folder, newest first, with the library each one was taken of. The buttons under the list:

Use selected	Make the highlighted snapshot this session's baseline.
New snapshot	Record every tag in the library now (also `s`). It reads the library and writes nothing to it.
Restore from this	Compose the command that would put this snapshot's tags back. See Undoing changes.

Delete selected	Delete the highlighted snapshot file, after asking (also `Delete`).

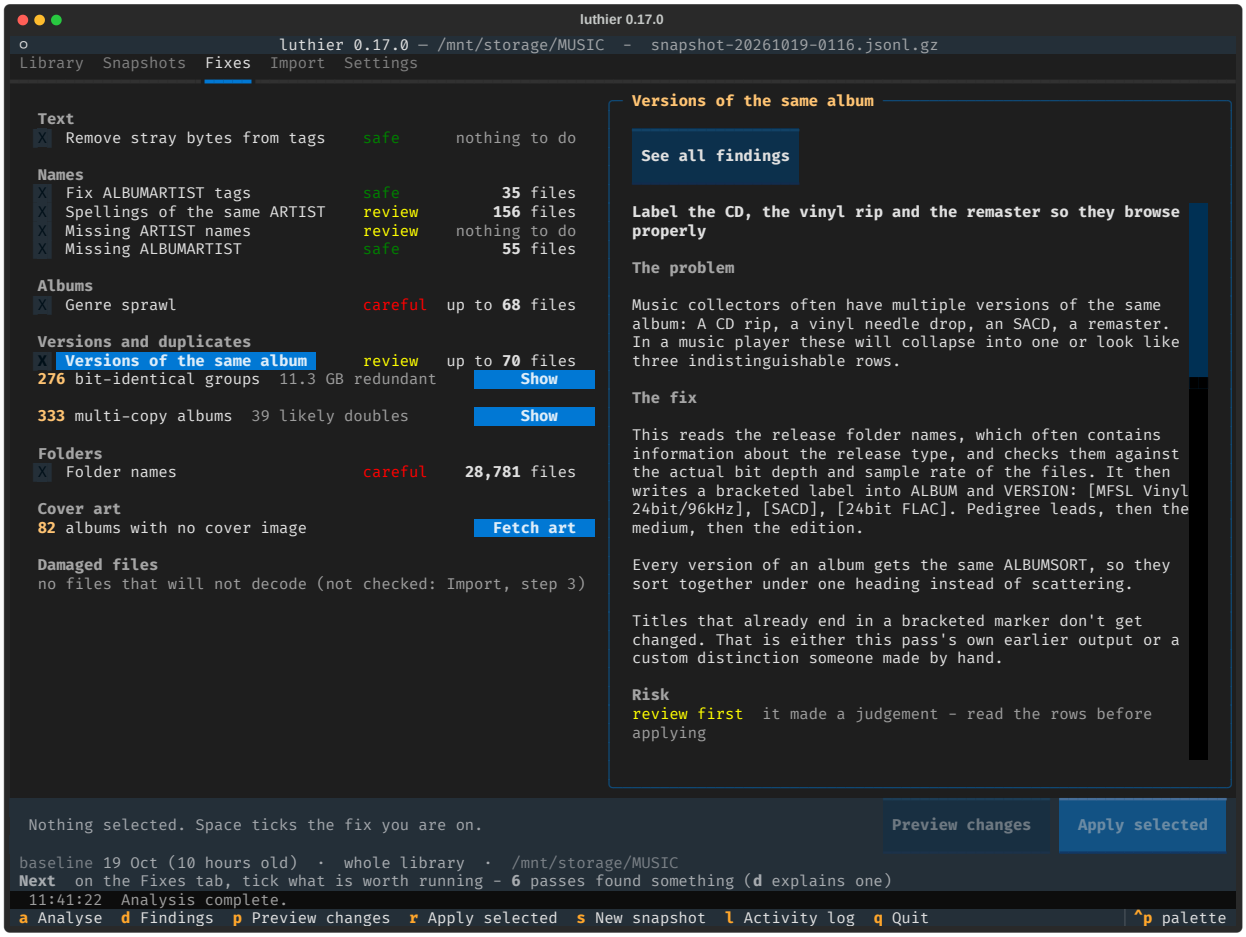
Baseline and restore points

One snapshot is the **baseline**: the one every fix is planned from. Planning from a snapshot rather than from the disk is what makes dry runs possible, since every count comes from the plan. The baseline is also the snapshot that goes stale as the library changes under it.

Every other snapshot is a **restore point**. Restore points never go stale: a restore only ever writes what that snapshot recorded.

A snapshot folder often holds snapshots of more than one library, for example the library and the drive it is mirrored to. The console only picks a baseline that was taken of the library you have open.

3.6 The Fixes tab



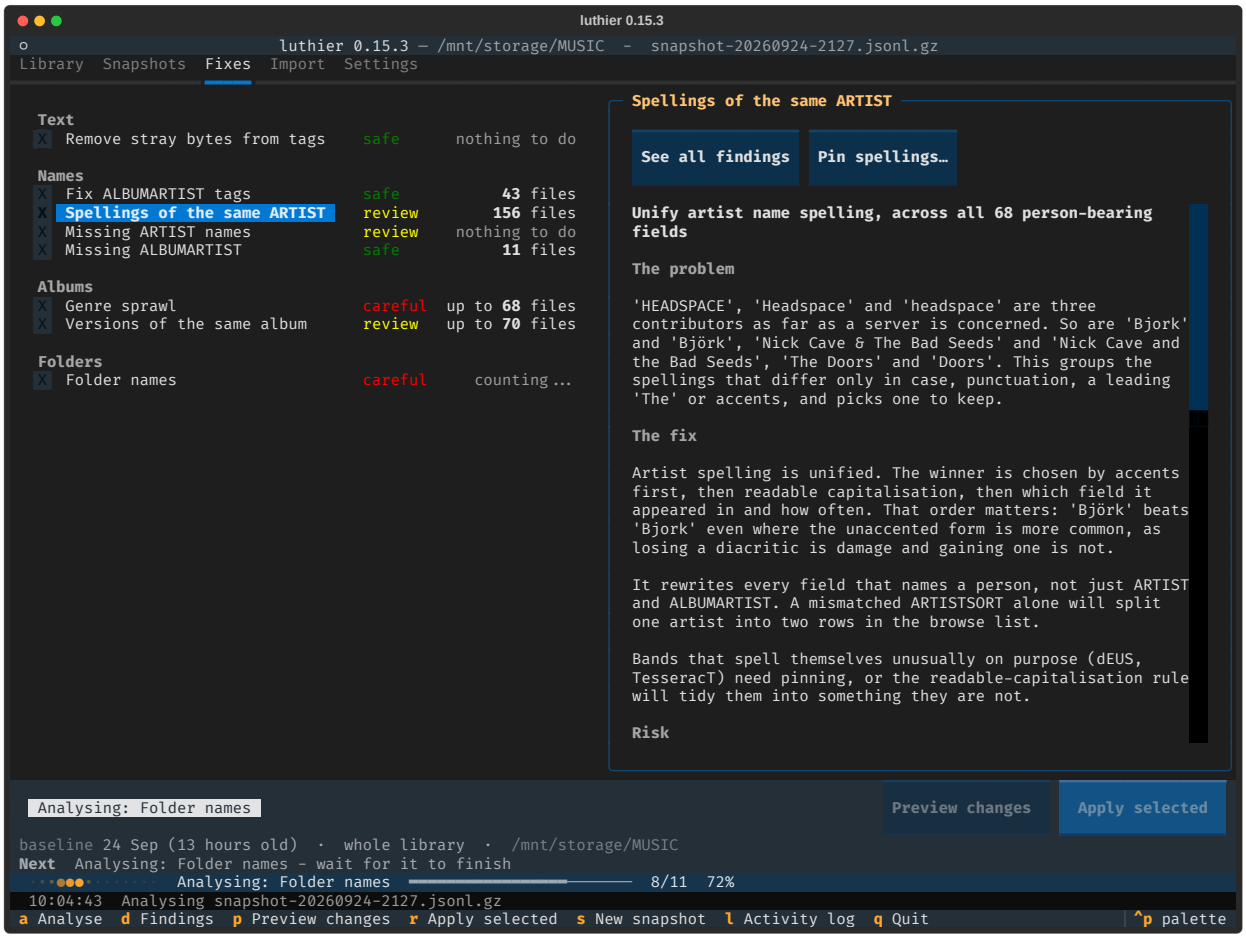
The Fixes tab. One row per fix on the left, grouped as Text, Names, Albums, Versions and duplicates, and Folders, with the findings no pass fixes in the sections they belong to. The pane on the right explains the highlighted fix, with its buttons along the top.

Every pass that can write is a row: a checkbox, how far to trust it, and how many files it would touch. Highlighting a row explains it in the pane beside it, in fixed sections:

The problem	What is wrong in the library, and why it matters.
The fix	What the pass does about it, and how it decides.
Risk	Its risk level, and what could go wrong.

Options	Switches that widen what the pass will do, on the passes that have any. All are off by default.
Impact	What it touches, what it writes, what it leaves alone, and every album, folder, field or rule it would change.

The buttons over the pane are **See all findings**, and, on the rows they belong to, **Pin spellings…** (Spellings of the same ARTIST) and **Undo last rename** (Folder names). Under the list, a bar adds up the selection (how many fixes, how many files, the highest risk, and the order they will run in) beside **Preview changes** and **Apply selected**.

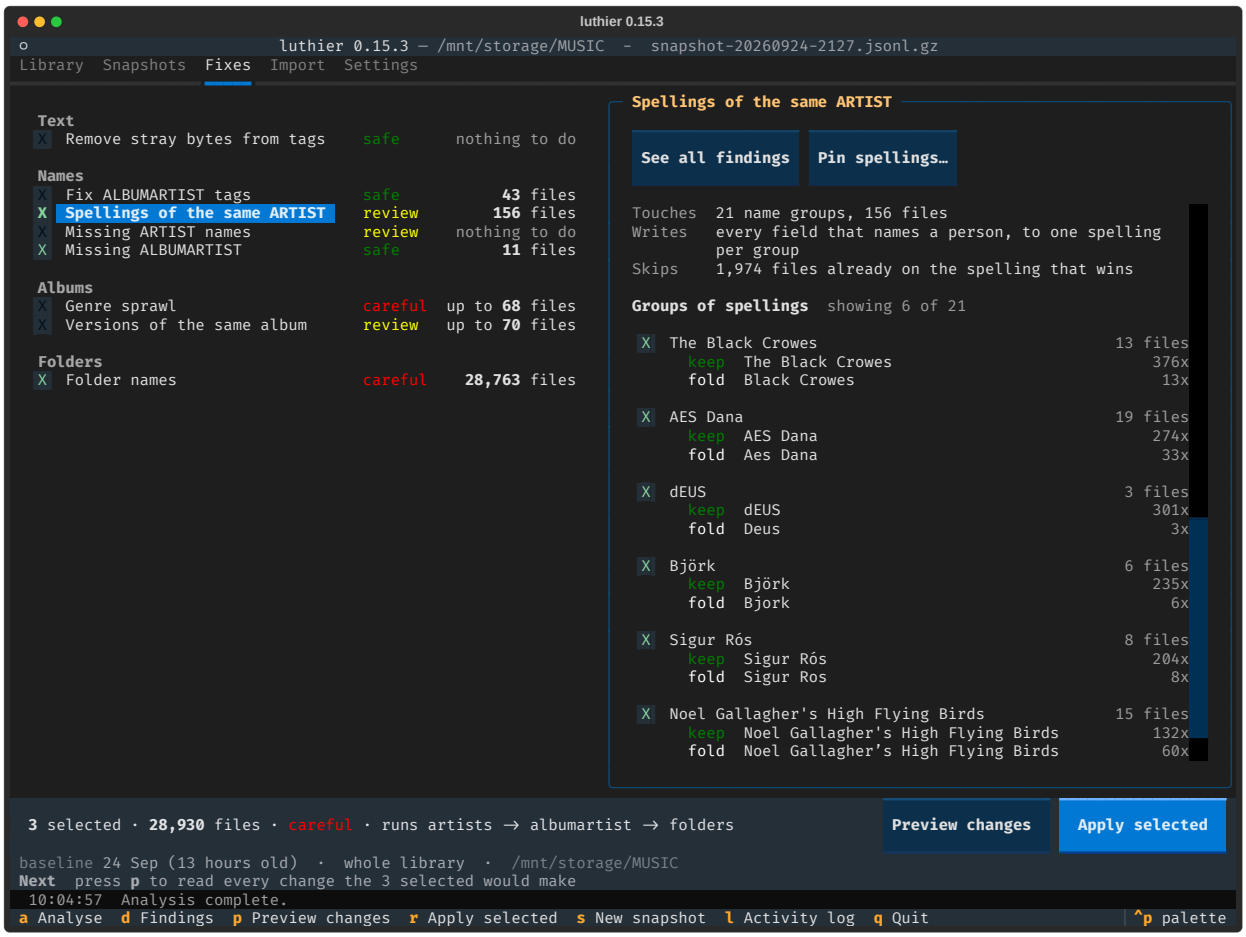


Analysing. The counts fill in pass by pass while the light sweeps the status line.

Risk levels

SAFE	Deterministic, and reversible from a snapshot. No judgement involved.
REVIEW FIRST	The pass made a judgement. Look at what it decided before applying.
CAREFUL	It overwrites values that were probably meant, or moves files. Read every row.

Impact, and unticking what a fix gets wrong



The Impact section. One card per album, each with a tick, the value it holds now and the value it would get.

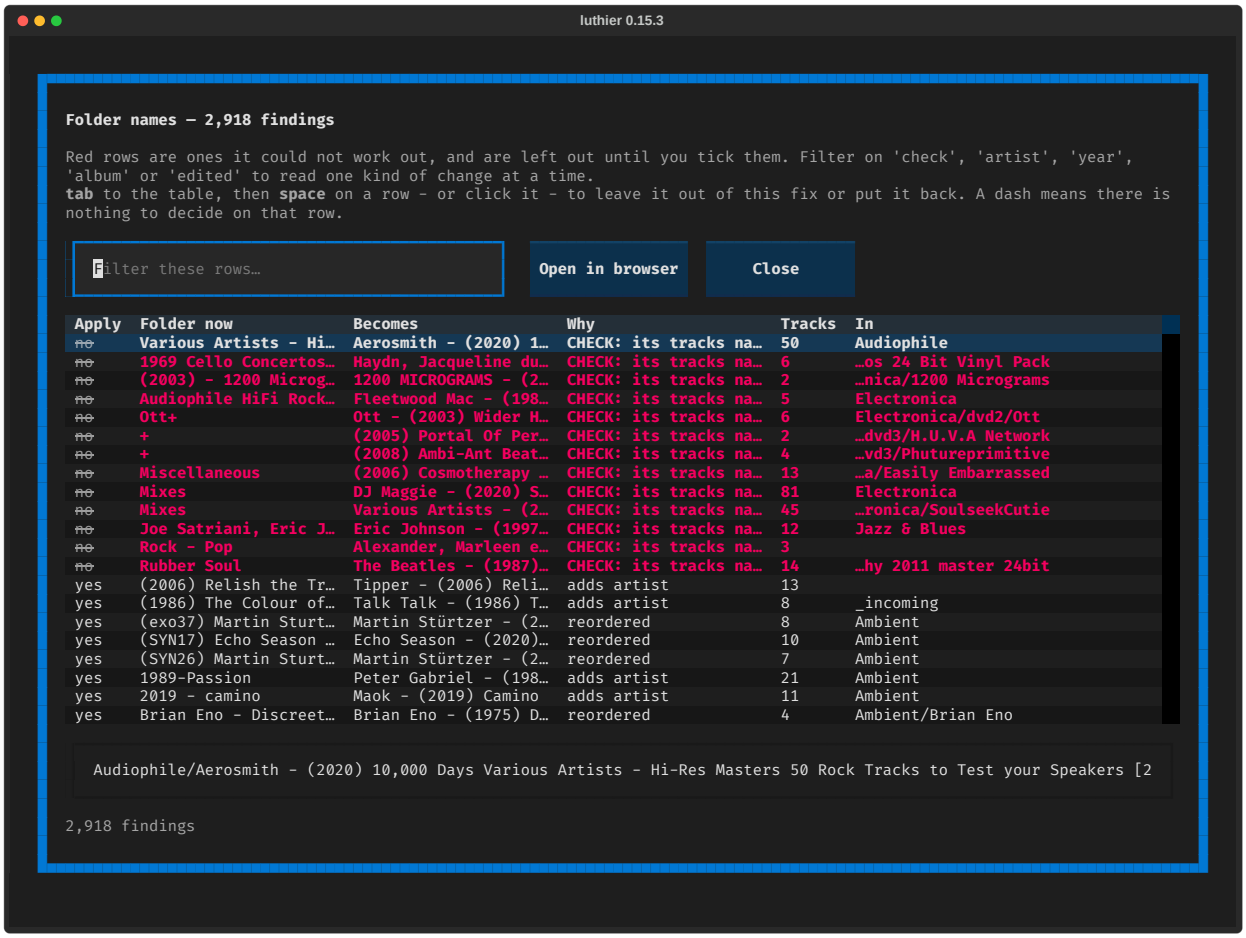
A fix is not all or nothing. Every item the Impact section lists has a tick, and so does every row behind it. Unticking one takes it out of the plan itself, so it leaves the count on the row, the preview and the write together: a pass that is right about forty albums and wrong about one can still be run on the forty.

What you untick is remembered by folder, field, alias or rule rather than by position, so it survives re-analysing and changing an option. Some passes decide at a coarser grain than they list at (`clean` per tag field, `aliases` per key, `genres` per rule, `editions` per album across its versions) and each says so above its list.

3.7 Reviewing: findings and preview

Nothing is applied unseen. Two views answer two different questions:

d **See all findings**	**Was this pass right?** Every row one pass produced, including what it looked at and *declined*: the genre no rule matched, the alias held back by a conflict, the tier left unticked. The **Apply** column shows what you have left out; `Space` puts a row back or takes it out.
p **Preview changes**	**What lands on disk?** Every write the whole selection would make, grouped by pass, in the order they run. Nothing a pass declined and nothing you unticked appears.



See all findings for Folder names. Folders it could not work out are red and left out until ticked; the highlighted row's new name can be edited below the table.

Both views can be filtered, and both open the same data as a sortable page in your web browser (`e` or **Open in browser**) when a thousand rows want more room than a terminal has. A table is capped at 3,000 rows; filtering still finds a row past the cap.

A preview looks like this. Invisible debris is made visible, because otherwise *before* and *after* would look the same: `␣` is a space, `\x00` a NUL byte and `\ufeff` a byte-order mark.

```
What apply would change in _incoming/ - 6 passes

  Remove stray bytes from tags       36 changes    safe
     comment  +1 more   ␣~ N ~        →  ~ N ~        _incoming/Facial Memories/01 …
  Fix ALBUMARTIST tags                79 changes    safe
     album artist      duplicate key  →  folded into ALBUMARTIST
  Spellings of the same ARTIST         6 changes    review first
     artist            SCANN-TEC      →  Scann-Tec    10 files
  Missing ALBUMARTIST                  7 changes    safe
     ALBUMARTIST       (none)         →  Norah Jones  _incoming/Norah Jones - 2004 …
  Genre sprawl                        23 changes    careful
     GENRE             Ambient; IDM   →  IDM          _incoming/(inre 045) Aes Dana …
  Versions of the same album          11 changes    review first
     ALBUM             Rumours        →  Rumours [SACD DSF]
```

Some rows are held back before you touch them. **Folder names** leaves out, in red, the folders it could not work out: tracks that name several different albums or artists, or a new name that would drop a word the old one had. Tick one to put it back in.

3.8 Applying

Press `r` or **Apply selected**. A confirmation spells out what will be written and how far to trust it; nothing happens until you confirm. While it runs, the status bar counts the files it has been through out of the total.

The order you tick fixes in does not matter. They always run in the only order that works:

```
clean → aliases → artists → filenames → fill → albumartist → mbfill → mbcheck → genres →
editions → folders
```

`clean` runs before anything that groups by name, `fill` before `albumartist` derives from what it wrote, and `folders` last, from the tags everything else has just settled.

After applying, take a new snapshot. The dialog that lists what was applied offers one (**Take a snapshot**), as does `s` later. The old baseline describes the library as it was, so the Fixes tab goes on showing the changes as still to do, and the session strip calls the baseline stale.

3.9 Stale baselines

Every fix is planned from the baseline, never from the disk. An out-of-date baseline therefore offers to redo work already done, and an album that arrived after it is in no plan at all: not proposed, and not reported as skipped. The session strip warns when:

- the snapshot is more than a day old;
- a folder in the library has changed since it was taken; or
- the library holds album folders the snapshot has never seen; or
- luthier has written to the library since it was taken.

The last three are evidence rather than age, and on start-up any of them also opens a dialog offering a new snapshot. **Not now** carries on with the old baseline. What luthier writes (fixes applied, folders moved out or restored) is recorded in `library-writes.jsonl` beside the snapshots, because a tag written in place leaves no trace a folder check could find.

```
 baseline 3 Sep (11 days old)  ·  _incoming/  ·  2 folders unseen  ·  /mnt/storage/MUSIC
 baseline is stale   Rock/Talk Talk changed after this snapshot was taken - press s for a new
one
```

Writing is stricter than the warning. Before a fix writes a file, the tags it reads and writes there are compared with the baseline; a file where they have changed, whether by hand or by a fix earlier in the same run, is left alone and counted in the log. Without that, **Fix ALBUMARTIST tags** would promote an album artist and **Missing ALBUMARTIST**, planned before it, would overwrite it. Take a new snapshot and apply again to pick those files up. The command line does the same with `--apply`.

When the snapshot describes another library, or there is none at all, **Preview changes** and **Apply selected** are disabled, with the reason on screen. The fix in every case is the same: press `s` for a new snapshot.

3.10 Undoing changes

Restore from this on the Snapshots tab composes the exact command that puts the highlighted snapshot's tags back: a dry run, a full restore, and the one-tag form that is usually what you want. The command is broken across lines so it pastes as one. The console does not run it for you: a restore's output matters line by line, including which files it refuses because their audio changed, and that belongs in a terminal you can scroll.

```
luthier restore \
    /music \
    ~/music-tag-backups/snapshot-20261015-0930.jsonl.gz \
    --only ALBUMARTIST --apply
```

Folder renames are the exception, because they are undone by moving folders rather than by writing tags. **Undo last rename** on the Folder names fix restores the last run, after listing what it will move. See Folder names. Folders moved out from the duplicate lists are restored the same way; see Moving copies out.

3.11 Other findings

Some problems are not the tool's decision. They sit on the Fixes tab under the passes they belong with, each a count and a button, and the Library tab sums them up with **Review** to jump there:

FINDING	SECTION	BUTTON	WHY LUTHIER STOPS THERE
Bit-identical groups	Versions and duplicates	Show	Which copy to keep is a judgement no rule should make.
Multi-copy albums	Versions and duplicates	Show	An SACD beside a vinyl rip is a collection, not a mistake.
Albums with no cover image	Cover art	Fetch art	Choosing a cover needs eyes.
Files that will not play	Damaged files	Show	Damage, not something to rewrite.

Both duplicate lists are listed by album, one row per folder holding a copy, because the question is which of two records to keep, not which of a thousand files. In the bit-identical list, **★ suggested keep** marks the copy with the better tags and the shorter path, and each other folder says how many of its files are held elsewhere. In the multi-copy list the best copy comes first, and every other copy is marked by how likely it is a double:

MARK	MEANING
★ best	The copy ranked best: lossless over lossy, then bit depth, sample rate, track count, a rip log or cue sheet, and tags.
≡ identical audio	Every track is bit-identical to a better copy. Safe to remove.
= same quality, size	Same tracks, format, bit depth and sample rate as a better copy, and within 2% of its size. Probably a double.
≈ same quality	Same format, bit depth and sample rate as a better copy, but a different size. Maybe a double, maybe another mastering.
○ other quality	No better copy of its quality. Usually deliberate, a vinyl rip beside a CD.

Moving copies out

In either duplicate list, `Space` on a folder (or a click) ticks it for moving out of the library, and **Move selected…** moves the ticked folders after listing them. Nothing is deleted: each goes into `.luthier-removed` at the library root, under a folder for that run and at the path it had, and a parent folder it leaves empty goes with it. Snapshots do not look inside `.luthier-removed`. Empty it yourself once you are sure.

Each list has its rule, and a tick that would break it is refused with the reason:

- **Bit-identical groups**: a folder can go only if every file under it, subfolders included, has a bit-identical copy that stays.
- **Multi-copy albums**: any copy can go, as long as one copy of the album stays and the folder holds that copy and nothing else.

A folder holding audio the snapshot never saw is left where it is. The moves are logged with the folder renames, and the Versions and duplicates section then shows **Restore**, which undoes the last run of moves. Take a new snapshot afterwards, and rescan the library on your servers.

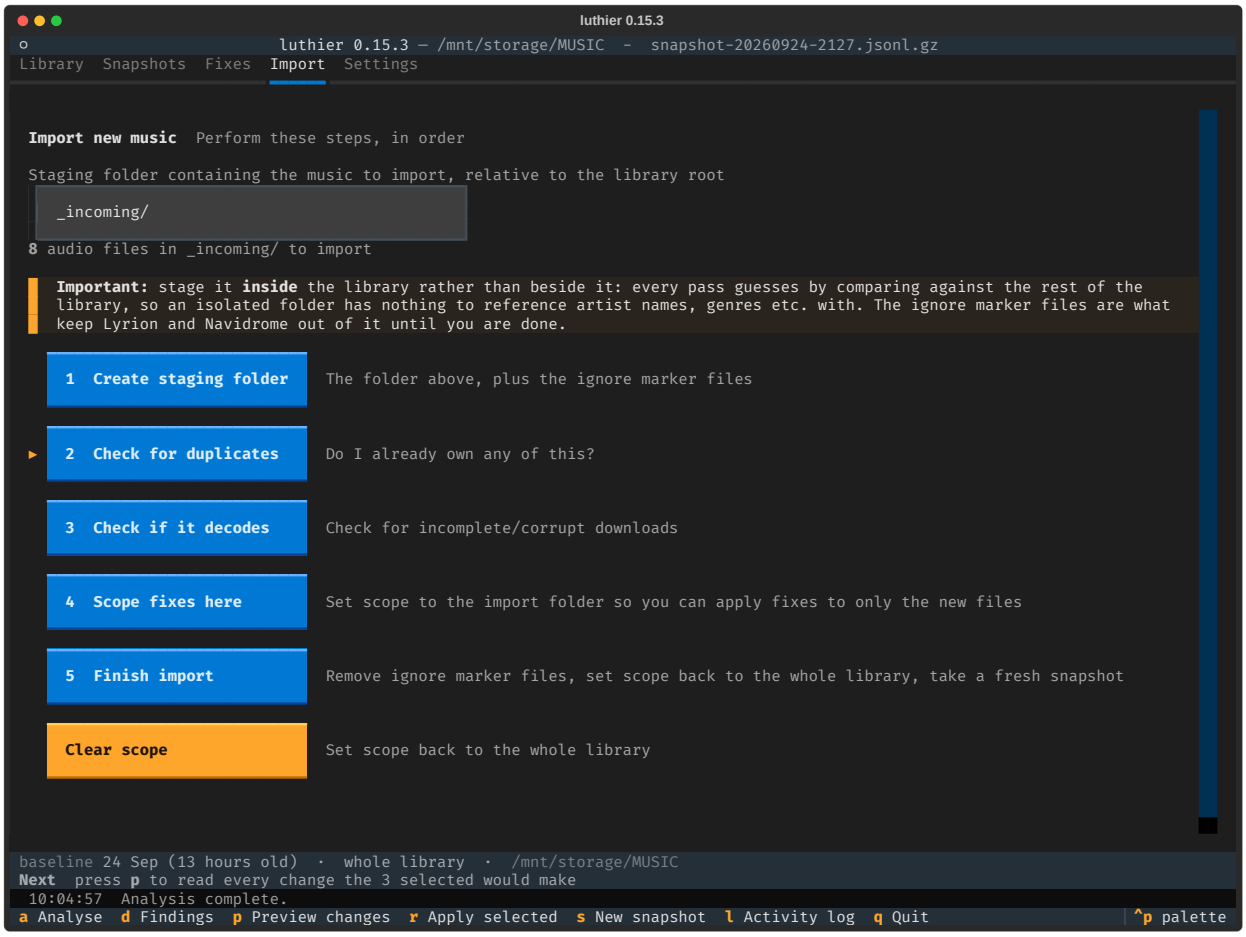
Fetching cover art

Fetch art opens a window listing the albums with no cover. Tick the ones to look up and press **Fetch artwork**: luthier searches the iTunes and Deezer catalogues and stages what it finds outside the library. **Preview fetched artwork** shows every candidate in your browser; it stays greyed out until something has been fetched. Untick anything wrong and press **Install selected**; only the ticked covers are copied in. An album that already has a cover image, in its folder or embedded in its files, is never touched.

Checking files decode

Files that will not play are found by reading the audio itself rather than the tags. That is the only way to catch a truncated download, whose header is usually intact and so looks healthy everywhere else. Run it from the Import tab (step 3) or with luthier verify.

3.12 The Import tab



The Import tab. Five numbered steps, in the order they are meant to run; an arrow marks the next one.

The passes that guess well do so by comparing against the whole library. Run on an isolated download folder they have nothing to compare with, and the new `Bjork` sits down next to your `Björk`. So new music is staged *inside* the library, in a folder the music servers are told to ignore, and only the fixes' writes are limited to it.

Type the staging folder, relative to the library (by default `_incoming/`), then:

1. **Create staging folder**: creates it with the `.lmsignore` and `.ndignore` marker files that keep Lyrion and Navidrome out until you are done. Move the new music into it.
2. **Check for duplicates**: do you already own any of it? Each album is reported as **identical**, **variant** or **new**; see arrivals.
3. **Check if it decodes**: finds incomplete and corrupt downloads, the commonest fault in new files and far cheaper to notice now than a year later.
4. **Scope fixes here**: limits every fix's writes to the staging folder, while each pass still reads the whole library. Then analyse, review and apply on the Fixes tab as usual.
5. **Finish import**: removes the ignore markers, sets the scope back to the whole library and takes a fresh snapshot. Rescan your music server afterwards.

Clear scope sets the scope back to the whole library at any time.

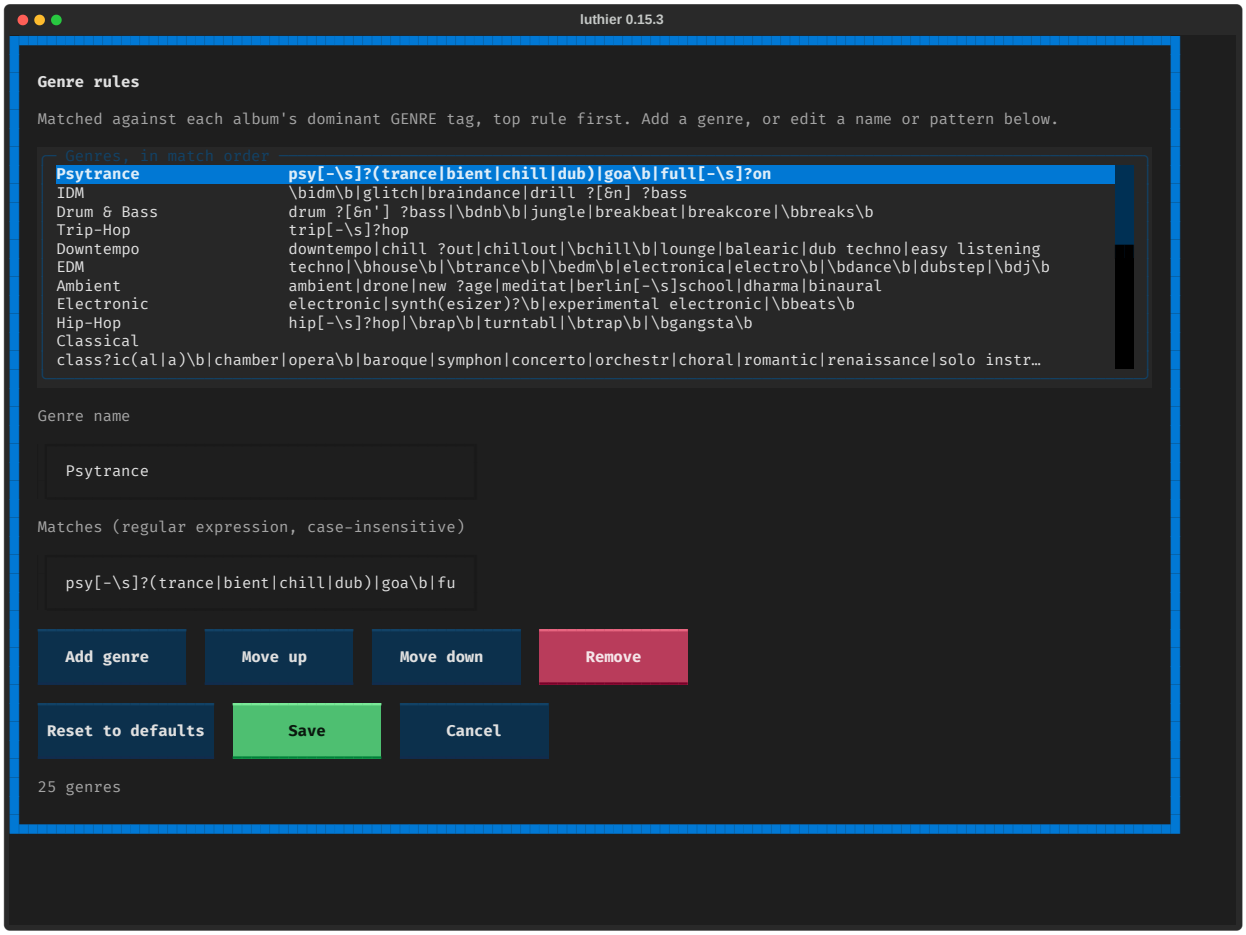
3.13 The Settings tab

Where the library is, where snapshots are kept, and the Import tab's incoming folder, saved to `~/.config/luthier/config.json` so none of it is typed again. Paths are saved absolute. A change made while a library is open takes effect the next time the console opens, rather than switching libraries under plans made for the old one.

Checking for updates

Once a day, when the console starts, it asks `phorel.net` whether a newer luthier is out, and says so if one is. It sends a random id made for this install, the luthier version and the name of the operating system, and nothing about your library. The id lets the number of installs be counted; no addresses are kept. Untick **Check for updates** and nothing is sent at all.

Editing the genre rules



The genre rules editor. The first match wins, so order is the design.

Edit genres… opens the rules the Genre sprawl fix folds free-text genres onto: an ordered list of genres, each with the pattern it matches. **Add genre** puts a new one at the top of the match order, where a broader rule already in the list cannot shadow it; **Move up** and **Move down** nudge it into place; **Remove** and **Reset to defaults** do what they say. **Save** writes `~/.config/luthier/genres.json`, which the command line uses too.

CHAPTER 4

The fixes in detail

Eight passes can write to the library. Each is a row on the Fixes tab and a command of the same name. They are listed here in the order they run.

On the command line every pass works the same way: give it a snapshot and the library root, read the dry run, then add `--apply`. `--only PREFIX` limits the writes to part of the library; see Command-line conventions.

4.1 Stray bytes · clean

Console: **Remove stray bytes from tags** Group: Text SAFE

Taggers leave debris behind: a NUL terminator written into the value by a C program, a zero-width space pasted from a web page, a byte-order mark, a trailing newline. None of it is visible and all of it matters: `Pink Floyd\x00` is a different artist from `Pink Floyd`, so one band shows up as two.

This pass removes the debris. Prose is treated as prose: a newline in a comment, lyric or rip log is content, so those fields keep their line structure and lose only the debris, while ordinary fields have their whitespace collapsed to single spaces. Nothing is rewritten into a value the tool thinks is better; it removes bytes, it does not improve wording.

```
luthier clean snap.jsonl.gz --root /music            # what is dirty
luthier clean snap.jsonl.gz --root /music --apply
```

4.2 Non-standard tag keys · aliases

Console: **Fix ALBUMARTIST tags** Group: Names SAFE

Vorbis comments have no registered name for the album artist, so taggers invented several: `ALBUMARTIST`, `ALBUM ARTIST` with a space, `ALBUM_ARTIST`, `ALBUMARTISTS`, `BAND`. Servers read most of them, so one file can present two different album artists and the same act appears twice. One real library carried 8,332 of these.

Each alias is compared with the `ALBUMARTIST` on the same file and dropped only when it says nothing new: the same name, the same name spelled worse, or a name to promote because `ALBUMARTIST` is empty. Anything genuinely different is a judgement call (`Various Artists` beside `Thievery Corporation` may be a mistake or a real distinction), so it is reported with its album and never rewritten. One unsafe alias holds back the whole file.

```
luthier aliases snap.jsonl.gz --root /music --apply
luthier aliases snap.jsonl.gz --conflicts --out conflicts.txt   # the judgement calls
```

4.3 Artist spellings · artists

Console: **Spellings of the same ARTIST** Group: Names REVIEW FIRST

`HEADSPACE`, `Headspace` and `headspace` are three artists as far as a server is concerned. So are `Björk` and `Bjork`, and `Nick Cave & The Bad Seeds` and `Nick Cave and the Bad Seeds`. This pass groups spellings that differ only in case, punctuation, accents or a leading “The”, and picks one to keep. A spelling that keeps its accents wins first, because losing a diacritic is damage and gaining one is not; then readable capitalisation; then the spelling used most.

LEVEL	FOLDS	RISK
`case`	capitalisation only	very safe
`punctuation`	also `&`/`and`, dots, underscores, diacritics	usually safe
`articles`	also a leading `The`	can merge different acts

It reaches every field that names a person, not just `ARTIST` and `ALBUMARTIST`. A server builds its artist list from composers, ensembles, original artists and sort names too, which is why two entries in a browse list can look identical and still be two artists: `Air` with an `ARTISTSORT` of `AIR` is a different contributor from `Air` with none. One real library had 68 such fields.

```
luthier artists snap.jsonl.gz --root /music --level case --list
luthier artists snap.jsonl.gz --root /music --all-fields --apply
```

Pinned spellings

Some names are spelled the way their bands chose: `dEUS`, `TesseracT`. Counting votes elects whichever mistake is commonest, and preferring readable capitalisation picks exactly the wrong one. A **pin** overrides the vote, even when the library is unanimous.

In the console, highlight **Spellings of the same ARTIST** and press **Pin spellings…** It shows the name groups and every spelling the library holds for each; choose one and press **Pin selected**, or **Unpin** to remove one. Pins are saved to `~/.config/luthier/pins.txt`, one spelling per line, and apply to every run, console or command line, with no flag. On the command line `--pin NAME` and `--pins FILE` add to the saved list for one run. Pins that match no name in the library are listed as such, so a pin that has stopped working does not look like one that has done its job.

4.4 Albums with no tags · filenames

Console: **Albums with no tags** Group: Names REVIEW FIRST

Some albums arrive with no tags at all. A server shows their tracks by file name, so they look tagged in a track list, but they have no album artist, genre or cover, and no other pass can work on them.

The names usually say what the album is:
`Amon Tobin - Bricolage/01 - Amon Tobin - Stoney Street.flac` gives the artist, the album, the title and the track number. This pass reads them for folders where no file has an artist, album artist, album or title tag, and only when every file name starts with a track number, no two the same. A year in the folder name becomes the `DATE`. The artist is written in the spelling your library already uses, and `GENRE` comes from another copy of the album in your library or, failing that, from what most of the artist’s albums are tagged.

MusicBrainz is then asked for a release by that artist and title with as many tracks, each within three seconds of its file. When one matches, its titles, album artist and original year are written instead of the guesses from the names, along with its release id. A file name is typed by hand: `Bitter & Twistd` becomes `Bitter & Twisted`. Answers are kept in `~/.cache/luthier/musicbrainz.jsonl`, so analysing again does not ask again. Without a connection the names are used as spelled.

An artist neither your library nor MusicBrainz knows is left out unless *Include artists nothing confirms* is ticked. A folder of untagged files this pass reads is not offered to `fill` as well.

4.5 Missing artists · fill

Console: **Missing ARTIST names** Group: Names REVIEW FIRST

An album with no artist tag is not merely untidy: a server has to invent something, and it invents it from the folder name. That is how `R.E.M` appears beside `R.E.M.` in a browse list.

This pass reads the candidate from the folder layout, but only writes it if the library already knows that name from properly tagged files elsewhere, and takes the spelling from the tags: the folder says `Cranberries`, the tag written is `The Cranberries`. A folder name that matches nothing, or matches two artists, is left alone. Only `ARTIST` is written; `ALBUMARTIST` follows from it in the next pass.

Option: *Include uncorroborated guesses* (`--include-folder`) also accepts names read straight out of an artist directory with nothing to check them against. Review those.

```
luthier fill snap.jsonl.gz --root /music --list    # every proposal, both tiers
luthier fill snap.jsonl.gz --root /music --apply   # corroborated only
```

4.6 Missing album artists · albumartist

Console: **Missing ALBUMARTIST** Group: Names SAFE

Servers group albums by album artist. Without it, a compilation shatters into one album per track and the artist list fills with noise. It is the commonest reason a good library browses badly. This pass derives it from the tracks in each album folder and gives every folder a verdict:

VERDICT	MEANING
`single`	One consistent ARTIST across the folder. Safe to write; written by default.
`majority`	One artist dominates and the rest are “feat.” variants. Option *Include albums with one dominant artist* (`--include-majority`).
`various`	Genuinely many artists. Option *Include compilations as Various Artists* (`--include-various`) writes `Various Artists` .
`no-artist`	Nothing to derive from. Always skipped.

A featured artist does not turn an album into a compilation: `X` , `X` , `X feat. Y` resolves to `X` . Folders under `Classical` are skipped, because there the performers differ by track while the album is one work.

```
luthier albumartist snap.jsonl.gz --root /music            # verdict per folder
luthier albumartist snap.jsonl.gz --root /music --apply    # writes the safe ones
```

4.7 Empty fields from MusicBrainz · mbfill

Console: **Empty fields from MusicBrainz** Group: Albums REVIEW FIRST

A tagged album can still leave out what nothing in its folder can supply: the MusicBrainz ids Lyrion and Navidrome group and match by, the original year, the album artist of a compilation, disc numbers. This pass finds the album's release on MusicBrainz and fills those in.

An album is a folder, or a folder of disc folders (`CD1`, `Disc 2`). It is looked up by its `ALBUM` and its album artist, or the artist most of its tracks name, or *Various Artists* when none does. As with filenames, a release counts only when it has as many tracks as the album, each within three seconds of its file, and it must also be credited to the artist asked for.

Only empty fields are written: `ALBUM`, `ALBUMARTIST`, `ARTIST`, `TITLE`, `DATE` (the original year), `TRACKNUMBER`, `DISCNUMBER` on a multi-disc release, and the release, release group and album artist ids. A value already set is never replaced. The ids are written along with the rest, but an album that lacks only the ids is not looked up: that is most albums in most libraries, and nothing is wrong with them. mbcheck adds the ids when asked. An album already tagged with a different release id is left alone and listed. MP4 files get the text fields only: their track numbers and MusicBrainz ids are typed values luthier does not write.

Looking up is not part of analysing. MusicBrainz allows one request a second and an album needs two or more, so a whole library takes hours. Analysing answers only from what has been asked before, and lists the rest as not looked up yet. Press **Look up on MusicBrainz** in the pane to ask about the albums in scope; set a scope first to do part of the library. The same button stops it, and every answer is kept, so the next run starts where this one stopped.

4.8 Check all against MusicBrainz · mbcheck

Console: **Check all against MusicBrainz** Group: Albums CAREFUL

Tags typed by hand or by an old ripper drift from the record: a misspelled title, a track artist that is not the one credited, tracks numbered out of order. This pass finds each album's release the way mbfill does and lists where the tags that are set differ from it.

`ALBUM`, `TITLE`, `ARTIST` and `TRACKNUMBER` are compared, and `DATE` when *Include years that differ* is ticked: a reissue's own year is often the one meant. With *Add MusicBrainz ids* ticked, the release, release group and album artist ids an album lacks are written too. Case, accents, punctuation, guest credits ("feat.") and anything in brackets, such as "(Remastered)" or an edition label, do not count as differences.

Every album starts unticked, and the row reads *none ticked* until you tick one. Applying writes the release's value over each difference on the albums ticked and nothing else. **Look up on MusicBrainz** asks about every album in scope, not only those with empty fields, so on a large library set a scope first. Answers are shared with `mbfill`, so an album asked about by one is not asked again by the other.

4.9 Genre sprawl · genres

Console: **Genre sprawl** Group: Albums CAREFUL

Libraries accumulate genre strings the way a shed accumulates screws. `Alternative`, `Alternative & Indie`, `Alt. Rock` and `AlternRock` all mean one thing and none of them agree. This pass folds them onto a small canonical list.

- **Per album, not per track.** That is how people browse. The album's most common genre string decides for the whole album.
- **Order is the design.** Rules are tried in order and the first match wins, so specific rules sit before general ones: `Psytrance` before `EDM`, `Progressive Rock` before `Rock`.
- **Nothing is guessed.** A string no rule matches is reported for review, and an album with no genre tag at all is left alone.

It overwrites the existing genre on every track of every mapped album, and rewrites `REM GENRE` in cue sheets (`--no-cue` to leave them). A snapshot holds tags only: a restore puts the genre tags back, not the cue sheets. If some of your genres are deliberate and unusual, add rules for them first: in the console with Edit genres… in Settings or **Genres…** on this fix, or on the command line:

```
luthier genres snap.jsonl.gz --root /music --list-unmapped     # what needs a rule
luthier genres snap.jsonl.gz --dump-rules > my-genres.json     # export, edit...
luthier genres snap.jsonl.gz --root /music --rules my-genres.json --apply
```

4.10 Versions of the same album · editions

Console: **Versions of the same album** Group: Versions and duplicates REVIEW FIRST

A library that keeps a CD rip, a vinyl rip and an SACD of one record shows three identical rows. What separates them was never tagged; it is in the folder name. A server's album record carries only title, year, artist and artwork, so a label becomes visible only by joining the title. This pass writes:

TAG	VALUE	WHY
`VERSION`	`MFSL Vinyl 24bit/96kHz`	the label, on its own
`ALBUM`	`title [label]`	the only field a browse list shows
`ALBUMSORT`	`title`	keeps the versions of one record together

Only albums held in more than one version are labelled, plus lone copies whose pedigree is the reason they were kept (an MFSL or PBTHAL transfer). Ordinary albums keep their titles exactly. A label is built from the source (`Vinyl`, `SACD`, `WEB` …), pedigree (`MFSL`, `PBTHAL`, `DCC` …), edition (`Deluxe`, `Mono`, a remaster year) and the format as read from the files. A remaster year moves into `DATE`, with the original kept in `ORIGINALDATE` (`--no-dates` to leave dates alone). Where two versions cannot be told apart it says so rather than inventing a label; those pairs are usually true duplicates.

```
luthier editions snap.jsonl.gz --root /music --review   # only what needs a decision
luthier editions snap.jsonl.gz --root /music --apply
```

NOTE

A run scoped with `--only` can only write `ALBUMSORT` on the versions in scope. If the album already existed in another version, follow it with an unscoped run.

4.11 Folder names · folders

Console: **Folder names** Group: Folders CAREFUL

Names every album folder `Artist - (Year) Album`, followed by whatever else the old name said. In an `Artist/Album` layout the artist is not repeated: under `Brian Eno/`, `Another Day on Earth` becomes `(2005) Another Day on Earth`.

```
  + artist            2007 Life² (CD)
                         → Asura - (2007) Life² (CD)
  + reordered         Mogwai - As The Love Continues - (2021) {WEB - FLAC}
                         → Mogwai - (2021) As The Love Continues {WEB - FLAC}
  ! folder says 2017, DATE says 2012: _incoming/(inre 055.2) Aes Dana - Pollen [2017] WEB
```

Three rules keep it safe enough to run in bulk:

- **Nothing is lost.** Artist, year and album are taken out of the old name only where they match word for word; everything else stays, in its own order, after the album. `(CD, UK)` and `[WEB]` survive. A name that writes the artist in a form that cannot be picked out (`ACDC`, `BBKing`) is skipped and reported.
- **The tags decide.** What gets added comes from the album's own tags, never from a lookup or another folder.
- **Disagreement is reported, not resolved.** A folder saying 2017 whose `DATE` says 2012 is listed, and the folder's own year is kept. `--no-year` neither adds a year nor moves one.

Discs and format variants inside an album folder (`Album/CD2`, `Album/88khz`) are left alone. Names are made safe for the filesystems a mirror drive is likely to use: `\ / : * ? " < > |` are replaced, so `AC/DC` cannot create a directory.

In **See all findings** every new name can be edited: type over the highlighted row's target. A target containing `/` moves the album as well. Typing a name for a red row ticks it back in.

> **CAREFUL**
>
> Renaming moves files. Playlists that list them break, Lyrion and Navidrome need a rescan (and may count a renamed folder as a new album, losing its play counts), and a mirror sync copies them again. Take a fresh snapshot straight afterwards: older snapshots still find the files through the rename log, but a new baseline is the plan's view of the library.

Undoing a rename

Every rename is logged to `folder-renames.jsonl` beside the snapshots, and `restore` and `diff` follow that log, so an older snapshot still finds the files. **Undo last rename** (or `luthier folders SNAPSHOT --root ROOT --undo --apply`) restores the last run of renames. A folder renamed again since, or whose old name has been taken again, is left where it is and reported; folders the renames created and the undo leaves empty are removed.

With `--root`, it also lists album folders that are on disk but not in the snapshot, which would otherwise be missing from the plan without a word:

```
    ? not in this snapshot, so not considered: Rock - Pop/Karnivool/In Verses
  4 folders on disk are not in this snapshot - take a new one to include them.
```

CHAPTER 5

Finding problems without fixing them

These commands only read and report. Nothing in this chapter deletes or edits anything, except installing cover art when you ask it to.

5.1 What state is the library in? · report

Tag coverage per field, and the answers to the two questions that matter most before a clean-up:

- **How many albums are missing `ALBUMARTIST`, and how many can be fixed mechanically?** A folder whose tracks all share one `ARTIST` needs no lookup.
- **Which files hold identical audio?** FLAC stores an MD5 of its decoded audio, so identical rips are found regardless of tags, file name or path, and at no extra cost.

```
luthier report snap.jsonl.gz
luthier report snap.jsonl.gz --json
```

5.2 Finding damage · verify

Tags live in the header, so a tag scan cannot tell a healthy file from one whose header survived but whose audio did not. `verify` decodes every file. FLAC is checked with `flac -t`, which recomputes the stored MD5 of the decoded audio, an exact check; everything else goes through `ffmpeg`.

```
luthier verify /music -o verify.jsonl --snapshot snap.jsonl.gz --jobs 4
```

Results are written as they are produced, and `--resume` skips files already recorded, so a scan measured in hours survives an interruption. Keep `--jobs` low on spinning disks: the limit is seek time, not processor.

5.3 Finding duplicates · duplicates

```
luthier duplicates snap.jsonl.gz --root /music --verify verify.jsonl
```

It separates three things that usually get lumped together:

- **Proven duplicates**: copies whose decoded audio is identical. One can go without losing anything.
- **Possibly duplicated**: same track count, format, bit depth, rate and almost the same size, but *different* audio. Never counted as reclaimable, because two different vinyl transfers look exactly like this.
- **Distinct masterings**: an SACD rip beside a vinyl transfer beside a CD rip. A collector holds these on purpose; `--redundant-only` hides them.

Within a group, copies are ranked best first with the reasoning shown: no unplayable files, lossless over lossy, greater bit depth, higher sample rate, a more complete track list, a rip log or cue sheet, better tags, then size. DSD is not treated as low fidelity.

5.4 Do I already have this? · arrivals

Asked while new music is still in the staging folder and deleting it is still easy:

```
luthier arrivals snap.jsonl.gz --only _incoming/ --root /music
```

identical	Every track's decoded audio already exists in the library. Certain, not inferred, even when the arrival is untagged or named differently. Only FLAC carries the checksum that proves it.
variant	You own this release, but the audio differs. It names the better copy and why; two copies nothing separates are reported as *equivalent*.
new	Nothing matches. File it.

5.5 Comparing a library with its mirror · compare

A backup drive drifts from the original: files removed on one side, tags edited on the other, a rip replaced by a better one. `compare` reports what is only on each side and what differs, so a mirror can be brought back into line deliberately rather than by a sync you have to trust blind.

```
luthier compare library.jsonl.gz mirror.jsonl.gz
```

5.6 Album art · artwork

Covers are looked up by artist and album name in the iTunes and Deezer catalogues. That works well for mainstream releases and badly for vinyl rips, bootlegs and anything with an edition suffix. Wrong artwork is worse than none, so nothing goes into the library directly: candidates are staged, reviewed on a contact sheet, and only what survives is installed.

```
luthier artwork snap.jsonl.gz --root /music --stage ~/art      # download candidates
luthier artwork --stage ~/art --review                         # contact sheet
#   ...delete the wrong ones from ~/art...
luthier artwork --stage ~/art --root /music --install --apply  # copy what survived
```

Albums are skipped when the folder already holds an image or the files carry embedded art; both Lyrion and Navidrome read embedded covers. On one real library that cut the candidates from 1,250 to 507.

CHAPTER 6

Command-line reference

Everything the console does is also a subcommand. The shape is the same throughout: take a snapshot, run a pass against it as a dry run, read what it would do, then run it again with `--apply`.

6.1 Conventions

- **Nothing writes without `--apply`.** Every pass is a dry run by default.
- **Passes read a snapshot and write to `--root`.** The snapshot says what the tags are; the root says where the files are.
- **`--only` means two things.** On the passes that write, `--only PREFIX` limits the writes to a path inside the library, relative to the root, while the pass still reads all of it. It may be repeated. On `diff` and `restore`, `--only TAG` names a tag field. Neither accepts the other sense, so a mix-up is an error rather than a surprise.
- `--skip PREFIX` ignores part of the library entirely, on top of the default skips.
- **Output is plain text,** one finding per line, so it pipes into `grep` and `less`. Several commands also take `--json`.
- `luthier *command* --help` lists a command's options. `luthier --version` prints the version.

6.2 Commands at a glance

COMMAND	WHAT IT DOES	WRITES
`ui`	the console (also plain `luthier`)	—
`snapshot`	record every file's tags	snapshot file
`diff`	what changed on disk since a snapshot	—
`restore`	write a snapshot's tags back	tags
`report`	library health from a snapshot	—
`verify`	decode every file, report unplayable ones	results file
`duplicates`	duplicated music, copies ranked by quality	—
`compare`	two snapshots, e.g. a library and its mirror	—
`arrivals`	do I already have what just arrived?	—
`artwork`	find, review and install album covers	cover images
`clean`	remove stray bytes from tag values	tags

COMMAND	WHAT IT DOES	WRITES
`aliases`	fold non-standard album-artist keys	tags
`artists`	collapse spellings of the same artist	tags
`fill`	fill a missing ARTIST from the folder layout	tags
`albumartist`	derive the missing ALBUMARTIST	tags
`genres`	unify genre tags onto a canonical set	tags, cue sheets
`editions`	tell versions of one album apart	tags
`folders`	name album folders from their tags	folder names
`skips`	the folders every pass ignores	—

6.3 Snapshots and restoring

`snapshot`

record every file's tags

```
luthier snapshot root -o OUTPUT [--exclude GLOB] [--jobs N] [--quiet]
```

`-o, --output FILE`	Snapshot to write (`.jsonl.gz`). Required.
`--exclude GLOB`	Skip paths matching this glob. Repeatable.
`--jobs N`	Files read in parallel (default 8). Helps on SSDs, much less on spinning disks.
`--quiet`	No progress output.

`diff`

compare files on disk against a snapshot

```
luthier diff root snapshot [--only TAG] [--limit N]
```

`--only TAG`	Only compare this tag. Repeatable; case does not matter. A field name such as `ALBUMARTIST` also matches what each format calls it (`TPE2` , `aART` , `WM/AlbumArtist`).
`--limit N`	Show at most N differences.

`restore`

write snapshot tags back to disk

```
luthier restore root snapshot [--apply] [--only TAG] [--progress] [--allow-content-changed]
```

`--apply`	Actually write. Without it, a dry run.
`--only TAG`	Restore only this tag, e.g. `--only ALBUMARTIST` , which also covers `TPE2` , `aART` and `WM/AlbumArtist` in other formats. Repeatable.
`--progress`	Print a running count, for long restores.
`--allow-content-changed`	Restore even files whose audio changed since the snapshot. Normally they are skipped.

6.4 Inspecting

`report`

library health report from a snapshot

```
luthier report snapshot [--min-tracks N] [--json]
```

`--min-tracks N`	Ignore folders with fewer tracks when judging albums.
`--json`	The same numbers as JSON.

`verify`

decode every file, report unplayable ones

```
luthier verify root -o OUTPUT [--snapshot FILE] [--exclude GLOB] [--jobs N] [--resume] [--quiet]
```

`-o, --output FILE`	Results file (`.jsonl`). Required.
`--snapshot FILE`	Verify the files listed in this snapshot.
`--exclude GLOB`	Skip paths matching this glob.
`--jobs N`	Parallel decoders (default 4). Keep low on spinning disks.
`--resume`	Skip files already in the results file.

`duplicates`

find duplicated music, rank copies by quality

```
luthier duplicates snapshot [--root ROOT] [--verify FILE] [--skip PREFIX] [--limit N] [--
  redundant-only] [--json]
```

`--root ROOT`	Library root, so rip logs and cue sheets count in the ranking.
`--verify FILE`	A `verify` results file, so damaged copies rank last.
`--redundant-only`	Hide albums deliberately held in several masterings.

`compare`

compare two snapshots

```
luthier compare snapshot_a snapshot_b [--skip PREFIX] [--limit N] [--json]
```

`arrivals`

do I already have the albums in my import folder?

```
luthier arrivals snapshot --only PREFIX [--root ROOT] [--verify FILE] [--min-tracks N] [--all]
  [--limit N] [--json]
```

`--only PREFIX`	The folder new music arrives in, e.g. `_incoming/`. Required.
`--root ROOT`	Library root, so rip logs count.
`--verify FILE`	A `verify` results file, so damaged copies rank last.
`--all`	List the new albums too.

`artwork`

find covers for folders that have none

```
luthier artwork [snapshot] --stage DIR [--root ROOT] [--review] [--html FILE] [--install] [--
  apply] [--list] [--delay S] [--limit N] [--skip PREFIX] [--only PREFIX]
```

`--stage DIR`	Staging directory for candidates. Required.
`--review`	Build the contact sheet (`--html` to choose where).
`--install`	Copy staged covers into the library; with `--apply`, actually copy.
`--list`	Just list the albums that need a cover.
`--delay S`	Seconds between lookups.
`--limit N`	Only process this many albums.

`skips`

the folders every pass ignores

```
luthier skips [PASS] [--args]
```

`PASS`	Add the extra folders this pass ignores, e.g. `albumartist`.
`--args`	Print them as `--skip` arguments, for a shell script.

6.5 The passes that write

All take a snapshot as their first argument, are dry runs without `--apply`, and accept `--skip PREFIX` and `--only PREFIX`. Most also take `--list` (just show what would change) and `--limit N` (show at most N). What each pass does is described in chapter 4.

`clean`

remove stray bytes from tag values

```
luthier clean snapshot [--root ROOT] [--apply] [--list]
```

`aliases`

consolidate non-standard album-artist keys

```
luthier aliases snapshot [--root ROOT] [--level LEVEL] [--apply] [--conflicts] [--out FILE]
```

`--level LEVEL`	How alike two spellings must be to count as one name: `case`, `punctuation` or `articles`.
`--conflicts`	Print the cases needing a human decision, with their albums.
`--out FILE`	Write the conflict report to a file.

`artists`

collapse spellings of the same artist

```
luthier artists snapshot [--root ROOT] [--level LEVEL] [--apply] [--list] [--pin NAME] [--pins
  FILE] [--all-fields]
```

`--level LEVEL`	`case` (safest), `punctuation` or `articles`.
`--all-fields`	Every field that names a person, not just ARTIST and ALBUMARTIST.
`--pin NAME`	A spelling to force for this run. Repeatable; with `--all-fields`.
`--pins FILE`	Another file of pinned spellings, added to the saved list.

fill

fill a missing ARTIST from the folder layout

```
luthier fill snapshot [--root ROOT] [--include-folder] [--min-tracks N] [--apply] [--list]
```

`--include-folder`	Also accept names taken from the folder alone, with nothing to corroborate them.
`--min-tracks N`	Tracks a name needs before it counts as known.

albumartist

derive the missing ALBUMARTIST

```
luthier albumartist snapshot --root ROOT [--apply] [--list] [--include-majority] [--include-
  various] [--majority SHARE] [--min-tracks N]
```

`--include-majority`	Also fix albums where one artist dominates.
`--include-various`	Also tag genuine compilations as Various Artists.
`--majority SHARE`	Share of tracks one artist needs to count as the album artist.

genres

unify genre tags onto a canonical set

```
luthier genres snapshot [--root ROOT] [--rules FILE] [--dump-rules] [--apply] [--list-unmapped]
  [--fallback GENRE] [--no-cue] [--min-tracks N] [--json]
```

`--rules FILE`	A JSON rules file. Default: the console's saved rules if there are any, else the built-in list.
`--dump-rules`	Print the rules that would be used, as JSON, and exit.
`--list-unmapped`	Show albums no rule matched.
`--fallback GENRE`	Give unmatched and untagged albums this genre instead of reporting them.
`--no-cue`	Do not rewrite `REM GENRE` in cue sheets.

editions

tell deliberate versions of one album apart

```
luthier editions snapshot [--root ROOT] [--apply] [--list] [--review] [--relabel] [--no-dates]
  [--no-albumsort] [--min-tracks N]
```

`--review`	Show only the albums needing a human decision.
`--relabel`	Also rewrite titles that already end in a bracketed marker, including this pass's own earlier output.
`--no-dates`	Do not move a remaster year into the release date.
`--no-albumsort`	Do not write ALBUMSORT; versions then sort by label.

folders

name album folders Artist - (Year) Album

```
luthier folders snapshot --root ROOT [--only PREFIX] [--skip PREFIX] [--no-year] [--undo] [--
  apply]
```

`--no-year`	Neither add a year nor move one.
`--undo`	Restore the last run of renames logged beside the snapshot.

`ui` — the console

```
luthier ui [root] [--snapshot FILE] [--snapshots DIR]
```

`root`	Library root. Default: the one saved on the Settings tab.
`--snapshot FILE`	Snapshot to plan from. Default: the newest one of this root.
`--snapshots DIR`	Where snapshots live. Default: the Settings tab's, or `~/.local/share/luthier/snapshots`.

6.6 Importing new music from the command line

The Import tab's workflow as a script. Each pass is still built from the whole library; `--only` holds only its writes to the staging folder.

```
mkdir -p /music/_incoming
touch /music/_incoming/.lmsignore /music/_incoming/.ndignore   # Lyrion, Navidrome

luthier snapshot /music -o pre-import.jsonl.gz
luthier arrivals pre-import.jsonl.gz --only _incoming/ --root /music
luthier verify   /music/_incoming -o import-verify.jsonl

S="pre-import.jsonl.gz --root /music --only _incoming/ --apply"
luthier clean       $S
luthier aliases     $S
luthier artists     $S --all-fields
luthier fill        $S
luthier albumartist $S
luthier genres      $S
luthier editions    $S
luthier folders     $S

rm /music/_incoming/.lmsignore /music/_incoming/.ndignore
luthier snapshot /music -o post-import.jsonl.gz
```

CHAPTER 7

Reference

7.1 Supported formats

FORMAT	SNAPSHOT	RESTORE
FLAC, Ogg Vorbis, Opus, Speex, Ogg FLAC, WavPack, APE, Musepack	full	full
MP3, MP2, DSF, DSDIFF, AIFF, WAV, TrueAudio (ID3)	text frames	text frames; other frames (cover art, ratings) are preserved but not backed up
MP4 / M4A / M4B	text atoms	text atoms only; cover art, freeform identifiers and typed atoms (track and disc pairs, flags, ratings) are left untouched
WMA (ASF)	text attributes	text attributes; pictures and identifiers are preserved but not backed up, and a number keeps its type when its value is unchanged

Embedded cover art and other binary tag data is deliberately not stored: it would dominate the snapshot's size, and no tagging tool removes it by accident. FLAC pictures live outside the comment block and are never touched by a restore.

7.2 How restores find files

Restores match files by path, and check them against the FLAC audio MD5 where there is one. A file whose audio changed since the snapshot is skipped rather than overwritten, unless `--allow-content-changed` is given. Retagging does not change that checksum, so a snapshot stays valid across any number of tag edits.

Paths are stored relative to the library root, so a snapshot taken of one copy of a library can be compared with a mirror of it on another drive. Folder renames made by luthier are followed through `folder-renames.jsonl`.

7.3 The snapshot file

A snapshot is gzipped JSON Lines: a header on the first line, then one record per file. It is meant to be easy to inspect with ordinary tools:

```
zcat snap.jsonl.gz | jq -r 'select(.tags.ALBUMARTIST == null) | .path'
```

Vorbis comment keys are case-insensitive and stored lowercased; matching a tag name with `--only` ignores case in every format. Path prefixes are matched exactly.

7.4 Folders the passes leave alone

Every pass reasons about albums, and none of that holds for audiobooks, project bounces or a player's waveform cache. Those folders are skipped, so a report describes records rather than everything on the disk:

```
$ luthier skips
Audiobooks
Backups
AutoSaves
Media
peaks
_broken
_partially_broken
```

Your own library's exceptions go in `~/.config/luthier/skips.txt`, one path per line, with `#` for comments. `luthier skips` marks them `# yours`. A skip names a whole folder, never a string prefix, so `Media` does not also hide a band called `Mediaeval Baebes`. `albumartist` also skips `Classical`; `luthier skips albumartist --args` prints the full list as arguments for a script.

7.5 Files and locations

PATH	WHAT IT HOLDS
`~/.config/luthier/config.json`	Library, snapshot and incoming folders from the Settings tab.
`~/.config/luthier/genres.json`	Your genre rules, from the console's genre editor.
`~/.config/luthier/pins.txt`	Pinned artist spellings, one per line.
`~/.config/luthier/skips.txt`	Your own folders for every pass to ignore.
`~/.local/share/luthier/snapshots`	Default snapshot folder, if none is set.
`snapshots``/folder-renames.jsonl`	The log of every folder rename, for restore and undo.
`~/.cache/luthier/`*`version`*	Where the single-file executable unpacks itself.

`~/.config` stands for `$XDG_CONFIG_HOME` when that is set.

7.6 Performance

Scanning is limited by disk seek time, not by the processor. On a spinning disk expect roughly 60 files a second, around ten minutes for a 40,000-track library; an SSD is many times faster. An interrupted snapshot has to be started again from the beginning; `verify` can be resumed.

7.7 Troubleshooting

SYMPTOM	WHAT TO DO
Preview and Apply are greyed out.	There is no snapshot of this library, or the baseline was taken of another one. The reason is on screen; press `s` for a new snapshot.
“baseline is stale”	The library has changed since the snapshot. Press `s`, then `a` to analyse again.
New albums are missing from every fix.	They arrived after the baseline. Take a new snapshot.
A fix is right except for a few albums.	Untick those in its Impact section or in See all findings, then apply.
Settings changes have no effect.	They apply from the next start. Quit and open the console again.
Files are “skipped” by the decode check.	No decoder is installed. Install `flac` and `ffmpeg`; see Optional tools.
The music server shows renamed albums twice.	Run a full rescan in Lyrion or Navidrome.
A restore skips some files.	Their audio changed since the snapshot. That is deliberate; `--allow-content-changed` overrides it.

APPENDIX A

Licence

luthier 0.21.0
Copyright © 2026 Phorel. All rights reserved.

luthier is freeware. You may use it free of charge, for any purpose, and pass on unmodified copies of it free of charge. Its source code is not published, and it may not be modified or reverse engineered. It comes without any warranty.

The components luthier is built from keep their own licences (Appendix B), and nothing in luthier's licence limits what those give you.

The full text of the licence follows. It is also included as the file `LICENSE`.

LUTHIER FREEWARE LICENCE

Copyright (c) 2026 Phorel. All rights reserved.

luthier ("the Software") is free of charge, but it is not open source. Its source code is not published. This licence sets out what you may do with the copy you received.

1. What you may do

You may install and use the Software, on any number of computers, for any purpose, private or commercial, without paying for it.

You may copy the Software and pass it on to others, provided that you pass on the complete and unmodified copy you received, including this licence and the third-party notices that come with it, and that you charge nothing for it beyond the cost of the medium.

2. What you may not do

You may not sell, rent or lease the Software, or include it in something you sell.

You may not modify the Software, or distribute a modified copy of it.

You may not decompile, disassemble or otherwise reverse engineer the Software, except to the extent that the law that applies to you permits it despite this restriction, or that a licence of a third-party component permits it for that component (section 3).

You may not remove or alter any copyright notice, licence or third-party notice in the Software.

3. Third-party components

The Software includes components made by others, each under its own licence. They are listed, with their licences and copyright notices, in the appendix "Third-party software and credits" of the luthier manual, and their licence texts are included with the Software. Those licences govern those components, and nothing in this licence limits the rights they give you.

TagLib is included under the Mozilla Public License 1.1. Its source code is available under that licence at https://github.com/taglib/taglib/tree/v2.3.2, and from the copyright holder on request at mail@phorel.net.

4. Ownership

The Software is licensed, not sold. The copyright holder keeps all rights not expressly granted here, including the right to publish later versions, or the source code, under different terms. Such terms do not change this licence for the copy you received.

5. No warranty

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. The Software changes the files it is pointed at; you use it at your own risk, and keeping a backup of your files is your responsibility.

6. Limitation of liability

TO THE EXTENT PERMITTED BY LAW, IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING LOSS OF DATA, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7. Termination

This licence ends automatically if you break any of its terms. You must then stop using the Software and destroy your copies of it. Sections 5 and 6 survive its end.

APPENDIX B

Third-party software and credits

luthier is built on the work of others. The components below are included in, or used by, the version you received. Each is the property of its authors and is used under the licence named; the required notices and licence texts follow the table.

Python packages

COMPONENT	VERSION	LICENCE	COPYRIGHT
Textual	8.2.8	MIT	© 2021 Will McGugan. The console.
Rich	15.0.0	MIT	© 2020 Will McGugan
markdown-it-py	4.2.0	MIT	© 2020 ExecutableBookProject; © 2014 Vitaly Puzrin, Alex Kocharin
mdit-py-plugins	0.6.1	MIT	© 2020 ExecutableBookProject
mdurl	0.1.2	MIT	© 2015 Vitaly Puzrin, Alex Kocharin; © 2021 Taneli Hukkinen
linkify-it-py	2.2.0	MIT	© 2020 tsutsu3; © 2015 Vitaly Puzrin
platformdirs	4.11.7	MIT	© 2010-202x The platformdirs developers
Pygments	2.21.0	BSD-2-Clause	© 2006-2022 by the respective authors
typing_extensions	4.16.0	PSF-2.0	© Python Software Foundation

Compiled into luthier

COMPONENT	VERSION	LICENCE	COPYRIGHT
TagLib	2.3.2	MPL-1.1 (or LGPL-2.1)	© Scott Wheeler and the TagLib contributors. Built unmodified into luthier's extension module; used under the Mozilla Public License 1.1. taglib.org
UTF8-CPP	4.2.0	BSL-1.0	© 2006 Nemanja Trifunovic. Used by TagLib.
pybind11	3.1.0	BSD-3-Clause	© 2016 Wenzel Jakob. Connects TagLib to Python.

Inside the single-file executable

COMPONENT	LICENCE	NOTES
CPython 3.11	PSF-2.0	© 2001-2023 Python Software Foundation. python-build-standalone distribution.
OpenSSL 3.5	Apache-2.0	openssl.org
SQLite	public domain	sqlite.org
zlib	zlib	© Jean-loup Gailly and Mark Adler
bzip2	bzip2 (BSD-style)	© Julian R Seward

COMPONENT	LICENCE	NOTES
xz / liblzma	0BSD	tukaani.org/xz
libffi	MIT	© Anthony Green, Red Hat, Inc. and others
ncurses	MIT-style (X11)	© Thomas E. Dickey, Free Software Foundation, Inc.
libedit	BSD-3-Clause	© The Regents of the University of California
mpdecimal	BSD-2-Clause	© Stefan Krah
Nuitka runtime	AGPL-3.0 with runtime exception	The exception lets compiled programs carry any licence. nuitka.net
Zstandard (decompressor)	BSD-3-Clause	© Meta Platforms, Inc. and affiliates

Programs and services luthier calls

flac (GPL-2.0-or-later; libFLAC BSD-3-Clause, Xiph.Org Foundation) and **FFmpeg** (LGPL-2.1-or-later / GPL, the FFmpeg developers) are run as separate programs when installed. They are not part of luthier and not included with it. Cover art is looked up through the **iTunes Search API** (Apple Inc.) and the **Deezer API** (Deezer SA); artwork remains the property of its owners.

This manual

Set in **Nimbus Sans** © (URW)++ Design & Development, 2014, under the GNU AGPL v3 with the URW font exception; **JetBrains Mono** © 2020 The JetBrains Mono Project Authors; the phorel wordmark in **Atkinson Hyperlegible Mono** © 2020 Braille Institute of America, Inc. The last two under the SIL Open Font License 1.1. Screenshots render in **Fira Code** © 2014 The Fira Code Project Authors, SIL Open Font License 1.1.

Licence texts

MIT License

Applies to Textual, Rich, markdown-it-py, mdit-py-plugins, mdurl, linkify-it-py, platformdirs and libffi, each under the copyright notices listed above.

```
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
associated documentation files (the "Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the
following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial
portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT
LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR
THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

BSD 2-Clause License

Applies to Pygments (© 2006-2022 by the respective authors, see its AUTHORS file) and mpdecimal (© Stefan Krah).

```
Redistribution and use in source and binary forms, with or without modification, are permitted provided
that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the
following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and
the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED
```

```
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```

BSD 3-Clause License

Applies to pybind11 (© 2016 Wenzel Jakob), libedit (© The Regents of the University of California) and the Zstandard decompressor (© Meta Platforms, Inc. and affiliates): the two conditions above, and:

```
* Neither the name of the copyright holder nor the names of its contributors may be used to endorse or
promote products derived from this software without specific prior written permission.
```

Python Software Foundation License Version 2

Applies to CPython and typing_extensions. Copyright © 2001-2023 Python Software Foundation; All Rights Reserved.

```
1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or
Organization ("Licensee") accessing and otherwise using this software ("Python") in source or binary form
and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a
nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display
publicly, prepare derivative works, distribute, and otherwise use Python alone or in any derivative
version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright
(c) 2001-2023 Python Software Foundation; All Rights Reserved" are retained in Python alone or in any
derivative version prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or incorporates Python or any part
thereof, and wants to make the derivative work available to others as provided herein, then Licensee
hereby agrees to include in any such work a brief summary of the changes made to Python.

4. PSF is making Python available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR
WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY
REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF
PYTHON WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON FOR ANY INCIDENTAL, SPECIAL, OR
CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON, OR ANY
DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or
joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF
trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or
any third party.

8. By copying, installing or otherwise using Python, Licensee agrees to be bound by the terms and
conditions of this License Agreement.
```

Other licences

TagLib is offered under the GNU Lesser General Public License 2.1 or the Mozilla Public License 1.1; luthier uses it under the latter. Both texts are installed with luthier, in `luthier/licenses/taglib/`, as are those of UTF8-CPP and pybind11. TagLib is included unchanged. Its source code is available under the Mozilla Public License 1.1 at github.com/taglib/taglib, and from mail@phorel.net on request. **UTF8-CPP** is under the Boost Software License 1.0 (boost.org/LICENSE_1_0.txt). **OpenSSL** is under the Apache License 2.0 (apache.org/licenses/LICENSE-2.0). **zlib**, **bzip2**, **xz** and **ncurses** are under their own permissive licences, which ask that their notices be kept and that altered versions be marked as such; their licence texts are included with their source distributions. **SQLite** is in the public domain.

Index

A

activity log 9
ALBUM ARTIST (tag key) 21
albumartist 23
ALBUMARTIST
  missing 23
aliases 21
--apply 29
applying fixes 16
arrivals 28
artist spellings 22
artists 22
artwork 28
  in the console 17

B

baseline 12
  stale 16
browser view 15

C

cache directory 7
Check for updates 20
Classical (skipped) 36
clean 21
command line 29
compare 28
compilations 23
config.json 20
console 9
  starting 9
contact sheet 18
cover art 28
credits 39

D

damaged files 27
decode check 18
diff 30
dry run 5
DSF 35
duplicates 27
  in the console 17
  removing 18

E

editions 25
empty fields 24

F

Fetch art 18
ffmpeg 7
filenames 22
files and locations 36
fill 23
findings table
  filtering 15
first run 7
Fixes tab 12
flac (program) 7
FLAC audio MD5 6
Folder names 25
folder-renames.jsonl 26
folders 25
formats 35
freeware 38

G

genre rules 20
Genre sprawl 24
genres 24
  editing rules 20

H

health report 27
held-back rows 15

I

Impact section 13, 14
Import tab 19
importing new music
  duplicate check 28
  from the command line 34
  in the console 19
installing 7
invisible characters 15

J

JSON Lines 35

K

keyboard shortcuts 10

L

Library tab 10
licence 38
luthier
  what it does 5
.luthier-removed 18

M

masterings 27
mbcheck 24
mbfill 24
mirror drive 28
missing ARTIST 23
moving copies out 18
MP3 35
MP4 / M4A 35
MusicBrainz 22
  comparing tags 24
  ids 24

N

non-standard tag keys 21
NUL bytes 21

O

--only 29
Other findings 17

P

performance 36
person fields 22
Pin spellings 22
pins 22
pins.txt 22
Preview changes 14
progress indicator 9

Q

quick start 7

R

renaming folders 25
  consequences 26
report 27
requirements 7
restore
  command 30
  from the console 17
  identity and safety 35
restore point 12
risk levels 13
--root 29
run order 16

S

safety model 5
scope 19
See all findings 14
session strip 9
Settings tab 20
  first run 7
--skip 29
skips 36
skips.txt 36
snapshot
  command 30
  file format 35
  in the console 11
  why not a filesystem snapshot 5
Snapshots tab 11
sort names 22
spinning disks 36
stale baseline 16
stray bytes 21

T

tag coverage 11
third-party software 39
troubleshooting 37
truncated downloads 27

U

undo 17
Undo last rename 26
unseen folders 26
untagged albums 22
unticking items 14
updates 20

V

verify 27
VERSION (tag) 25
versions of an album 25

W

WMA 35
wrong tags 24

X

XDG_CONFIG_HOME 36