Niente

The FLAC Integrity Checker

by Howard Rogers

Version 2.0 – November 2021

Niente User Manual
by Howard Rogers
Copyright © 2021 Howard Rogers

Table of Contents

1.0 Introducing Niente

Niente is a FLAC integrity checker, useful for those with large digital music collections to manage and curate.

It is simple to operate and completely non-destructive to the music files it catalogues and examines. The results of its examinations are written only to a database, for later reporting. At no time does Niente attempt to fix up any of the problems it detects: it leaves that to you, who knows your music collection and the way it should be organised the best.

Niente can do differential integrity checks (i.e., "Only check the files that I haven't checked within the past 30 days"), or it can do full checks (i.e., "Check everything, no matter how recently it might previously have been checked").

Three different sorts of integrity report can be produced, either as text files or displayed on-screen:

- audio stream validity

- recording year consistency

- performer name consistency

The first of these is a measure of *physical* integrity of the file: has the audio data changed since the file was first produced as a FLAC. The other two are measures of *logical* integrity: when you were tagging up your music files, did you tag things consistently in the first place? If the first report lists anything at all, you will probably need to restore from a good backup or re-rip the file from CD afresh. If the other two reports list anything, you can fix up your existing FLACs with a little bit of re-tagging effort.

I wouldn't say Niente does anything particularly revolutionary -but it does put you in full curatorial control of your own music collection, which is no bad thing!

1.1 Some Prior History

I wrote an earlier FLAC integrity checker, called the Absolutely Baching Flac Checker (or AFC for short) in 2019. It had exactly the same goals as Niente: to check the physical and logical integrity of my music collection.

It fulfilled that goal for quite a long time -but at a cost. AFC wrote the results of its check out to text log files, which was fine, though cumbersome to read after the event. But it also *modified the FLAC files themselves to 'stamp' them with the date of their last integrity check*.

Modifying the thing whose integrity you purport to vouch for is, frankly, a bit of a no-no! I should never have really put up with that 'feature', but it seemed a reasonable price to pay for peace of mind, so I ran with it.

Unfortunately, writing to files you care about has consequences: in this case, those files now looked 'new' to my differential file backup routines. That wasn't particularly a problem when it merely triggered a fresh backup from my PC's hard disk to my NAS Server's hard disk array in the loft. A few hours' disk and network activity each night is not exactly a struggle.

But I've recently begun uploading a backup of my NAS to a cloud storage service: and having my entire music collection require re-transferring up to the cloud, when my upload Internet connection runs at around 10Mbps on a good day... that's definitely *not* a great idea! It basically means my entire upload bandwidth is going to be permanently in use by a single program, which is not ideal!

Thus, the idea of replicating AFC's core functionality, but without the integrity checks causing the music files to be altered in any way, and thus *not* triggering a cascade of differential backups... that sounded like a good idea! And thus Niente was written to replace AFC, and do its work fundamentally better and less disruptively.

1.2 The Name

If you are a classical music fan, you will already know that *niente* means 'a gradual dying away of volume until only a bare whisper can be heard'. Reasonable people might wonder the relevance of such a name to a FLAC integrity checker!

Well: the idea is that Niente gets out of your way, makes no alteration to your music files, disappears into the background, makes itself barely noticeable.

It's why the 'mascot' for the program is the Cheshire Cat, a carving of which is to be found high up in the ceiling of Christ Church Cathedral, Oxford ...where Lewis Carroll was once a Mathematical Lecturer (and which you can see at the left).

The Cheshire Cat was famous for gradually fading away to nothingness, leaving behind nothing but his smile... and the idea is that Niente will similarly fade away into the background, leaving you with a faint smile of satisfaction that you *know* your music collection is physically uncorrupted and logically consistent.

You will need to get used to the Cheshire Cat in all his guises, as he will be used relentlessly to represent this fundamental idea underpinning the program's design and purpose!

1.3 What does Niente do?

When FLAC files are first created, the FLAC encoder will compute a 'hash value' for the audio signal component of the file. Think of a hash value as being like a 'checksum': a value which represents the result of performing assorted mathematical computations against an input. Importantly, the input in this case is *only* the audio signal component of the file: you can change the album art all you like; tag the thing with various versions of the composer's name or the performers involved... all such alteration of the non-musical parts of a FLAC will **not** alter the content of the audio signal component. Accordingly, the hash value computed from that audio signal should be invariant over time.

Unfortunately, files can suffer from 'bit rot' over time: they might be read incorrectly from disk (one tiny flaw in the magnetic materials involved might do it, for example), and then written back to disk -

now with a tiny flaw in its internal data. Or cosmic rays can affect data at rest on the hard disk (honestly: it happens!)… and so on and so forth.

So, whilst it should be possible to re-read the audio signal in a file, recompute the hash value and come up with precisely the same answer as the FLAC encoder originally calculated… sometimes (rarely, but definitely with a non-zero probability) the freshly-computed hash value for the audio signal *won't* match the one originally computed by the FLAC encoder at the time of the file's creation.

When such a mismatch happens, it means the audio signal has, for whatever reason, become internally corrupt. Sometimes, you'll be able to hear this corruption -as an audible 'click' or 'jitter' on the music as its being played. Sometimes, the corruption might mean the file becomes completely unplayable. But more often, the corruption will not be detectable by your ears and will lie there quietly, perhaps getting worse over time.

Niente spots these corruptions, however inaudible they might be, by re-computing the audio signal's hash value and comparing it to the original one the file's encoder wrote into the file at the time of its creation. A single bit-flip will make the two hash values wildly different and a simple textual comparison of the two will tell you which files have internal problems.

There's no simple fix for corruption when its detected, by the way: you either have to restore the file from a known-good backup, if you have one (which you should!). Or, you have to rip the file from its original CD once more and obtain a fresh, good copy with which to replace the damaged original.

Niente will compute hash values for you and use them to *detect* audio signal corruption, in other words; but it cannot fix such corruption when it finds it. That's entirely down to you!

Niente also does a couple of other checks whose usefulness depends on how you tag your FLAC files. If you follow my *Axioms of Classical Tagging*, for example (see https://absolutelybaching.com/music-articles/the-axioms-of-classical-music-tagging/), then you will:

- store the recording year in the ALBUM tag
- store the recording year in the DATE tag
- store the performers of the recording in the COMMENT tag
- store the principal or 'distinguishing' performer in the PERFORMER tag

For example, the Solti recording of Wagner's *Das Rheingold* would, under my axioms, be catalogued as:

- ALBUM=**Das Rheingold (Solti - 1958)**
- DATE=**1958**
- COMMENT=**Georg Solti, Vienna Philharmonic Orchestra, Oda Balsborg (Woglinde), Hetty Plümacher (Wellgunde), Ira Malaniuk (Flosshilde), Gustav Niedlinger (Alberich), George London (Wotan), Kirsten Flagstad (Fricka), Claire Watson (Freia), Walter Kreppel (Fasolt), Kurt Böhme (Faffner), Ebergard Wächter (Donner), Waldemar Kmentt (Froh), Set Svanholm (Loge), Paul Kuen (Mime0, Jean Madeira (Erda)**

and

- PERFORMER=**Georg Solti**

There are good reasons for tagging things this way, which I needn't get into now -but the problem is that we now have duplicated data about the place... and no mechanism by which we can ensure that the data that appears in the second place is 'in synch' with the same data as it appears elsewhere.

The '1958' in the ALBUM tag needs to match the '1958' in the DATE one, for example; and the 'Georg Solti' in the PERFORMER tag needs to match something (usually, but not invariably, at the beginning) in the COMMENT tag. There are no automatic ways of ensuring this level of logical *data consistency*, but Niente will at least check all these tags and make sure that they make sense -and produce reports identifying any digital files which 'break the consistency rules' in one or more ways.

Summing up, then: Niente will check for *physical* corruption within a FLAC file; it will also check for *logical* inconsistencies in the way you have tagged up your music files when first cataloguing them.

It is possible that you have devised your own way of tagging your music files, where the logical consistency checks Niente performs would not make sense. That's fine too: you don't have to produce the reports about logical inconsistencies if you don't need them. But everyone can at least take advantage of Niente's ability to check for the physical degradation of FLAC files.

2.0 Prerequisites for running Niente

There are some obvious things you need to do before you can successfully run Niente to check your music files. You must, for example, be using Linux. You also need to be able to run sudo commands when needed (describing how to achieve that on various distros is discussed in **Appendix B** of this document).

You must also store your digital music files in the lossless FLAC format: Niente can only check FLAC files and will not (by design!) even see any APE, WAV, MP3, OGG or any other digital music format.

There are some more specific software prerequisites too: you'll be told about them when you first try to run Niente on a system where they have not been installed previously. Probably the most important component is **ffmpeg**, which is available for all Linux distros (but may require some finagling to install on some distros, such as openSUSE and Fedora, which seem to think it might be patent encumbered and thus don't include it by default in their 'standard' repositories). In any case, as I say, the program itself will warn you about necessary software dependencies when it is first run.

2.1 Niente and the Axioms of Classical Tagging

The check for physical internal integrity that Niente performs on FLAC files can be exploited no matter the state of your digital music collection: a FLAC's a FLAC, no matter how careless tossed onto a file system it might be!

But the logical checks on your FLAC tags that Niente performs will only make sense if you are cataloguing and tagging your files in the way I describe in my article on the 'Axioms of Classical Music Tagging', available at https://absolutelybaching.com/music-articles/the-axioms-of-classical-music-tagging/. That outlines and explains in detail how you should tag your classical music files -and why.

In particular, Axioms 3 (governing what goes into the COMMENT tag), Axiom 5 (describing the need for an 'extended album name' to be stored in the ALBUM tag, Axiom 6 (what the PERFORMER tag is for) and Axiom 7 (guidelines for how to use the DATE [or YEAR] tag) need to be followed for Niente's logical checks to make any sense at all.

If you do not personally agree or follow those axioms, that's fine: Niente is, as I say, still of use for the physical integrity checks it performs; and there's no requirement for you to ever see the results of the logical checks it does that are of no interest to you.

I would recommend the Axioms as good guidelines to follow for an excellently-curated classical music collection, though, because they describe a coherent and highly functional way of doing the job. But Niente will still be of value to you, even if you're not collecting classical music, let alone whether you agree with my particular tagging recommendations if you are!

3.0 Installing Niente

To install Niente on your Linux system, simply type these commands at a terminal prompt:

```
cd
wget https://absolutelybaching.com/abc_installer
bash abc_installer --niente
```

The **cd** makes sure what is about to happen takes place in your own home folder. The **wget** command fetches the Absolutely Baching Installer script (and obviously needs to have been installed before you start: it's present by default on most Linux distros with which I'm familiar, though, so should generally work 'out of the box'. The **bash** command then runs that script.

The installer script will fetch the Niente software from the Absolutely Baching website and will copy it into the appropriate folders for you. Since root permissions are needed to do this copying, the script will prompt you to supply your sudo password at one point: you do need to be able to run with root permissions via sudo for the installation to succeed.

An optional step is now to **make sure that your terminal of choice always opens with dimensions of at least 102x26**. Most distros seem to default their terminals to opening at 80x24, and this will result in garbled output from Niente. So you should alter the 'profile' governing a terminal's default size. On Konsole, that's *Settings → Edit Current Profile*, but you also need to do *Settings → Configure Konsole* and turn **off** the 'Remember window size' option, so that the new profile values take effect. In Gnome, it's just click the terminal's 'hamburger menu' (three horizontal lines, up in the top left), then click *Preferences*. The default profile is named under the 'Profiles' item on the left, and can be altered there.

I say that it's an optional thing to change your terminal size in that Niente will work perfectly well without you doing it: but it is written to expect a 102x26 window and will be mostly unreadable garbage if you don't accommodate it in this way!

So, with the software installed and your terminal configured to be the 'right' size, you can run Niente for the first time by typing the simple command:

```
niente
```

It is quite likely that at its first run, Niente will simply display a message saying that you're missing some software prerequisites. Once you've installed them, though, then running Niente with this most basic of commands will prompt the program into doing not much more than displaying some very high-level and simplistic help on the sort of command-line parameters you can add to the command to get it to do meaningful work. Mastering those command-line parameters is how you get Niente to do serious, productive work for you, as we'll see.

Let me now walk you through the 'first run' of Niente in a bit more detail, with some screenshots!

4.0 Niente – First Run

When first run, Niente will scan your system for the presence (or absence!) of necessary software prerequisites. If it finds any software missing, it will prompt you to install it, and then quit. You can then install the software using the command it suggests (or via any other software installation mechanism with which you are more comfortable), and then re-run Niente, until it declares itself happy with the software environment it finds itself in.
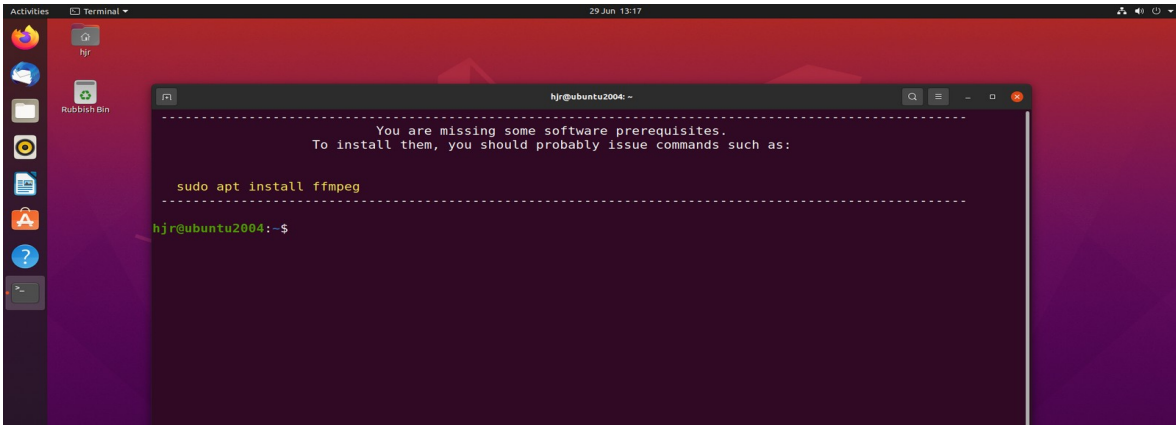

Here's a worked example on a freshly-installed (and totally default) Ubuntu 20.04.2 system:

You should be able to make out that the program has determined that, on this occasion, a single piece of 'missing' software has been detected: in this case, ffmpeg.

Notice that Niente is 'distro-aware': it knows it's being run on Ubuntu, so it's suggesting a **sudo apt install** command to get the required software installed. On openSUSE, it would provide a **zypper** command; on Fedora a **dnf** one, and so on. Niente has been tested on all major distros (anything in the top 10 of Distrowatch, basically), so it should get most of them right, but if it is being run on something out of the ordinary, it may not entirely understand how to install required packages -in which case, it's down to you to understand your choice of distro's package management processes!

Anyway: clearly, this particular distro needs me to install some extra software, so let me run the command shown and then re-run Niente afterwards:
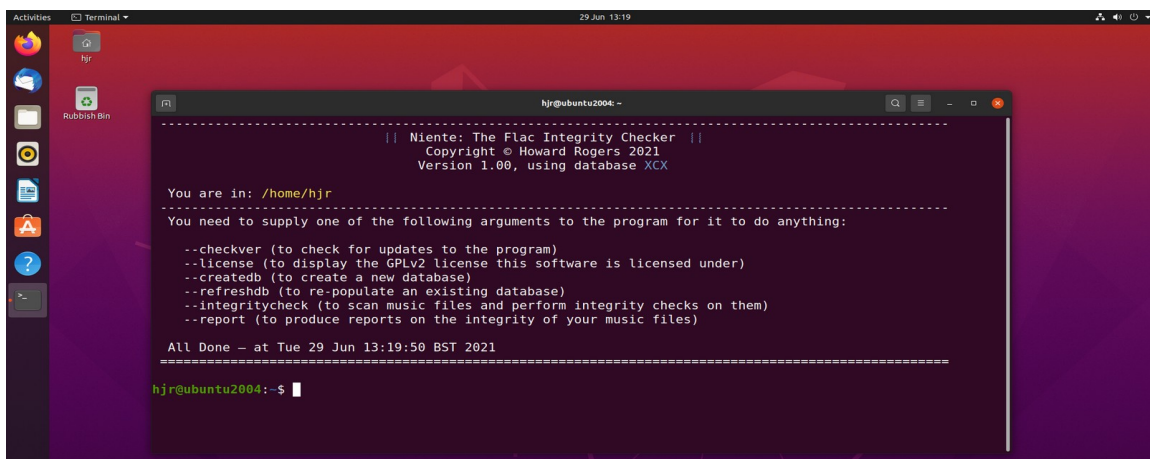
Well: it's certainly a different response -but it frankly isn't very exciting! It's telling you some of the 'run-time parameters' you can supply to the program, which will actually make it do useful things. These are all detailed at great length under **Section 8** of this document.

But leave those aside for now: the point is that Niente has been installed, its software dependencies have been met, and it's able to run. That's progress, at least!

5.0 The Niente Workflow

To do anything productive with Niente, you have to do four main things in sequence:

- Create a database (you only do this once)

- Populate the database with details of what music files you have (called "refreshing the database")

- Scan your music files (called 'doing an integrity check')

- Report on the results of the integrity check

As mentioned, you only create a database once (though there's nothing to stop you deleting it and creating it again; or creating lots of different databases to record different parts of your music collection -if, say, your classical music is stored somewhere distinctively different from your non-classical stuff and you want to keep their integrity checks distinct from each other).

The database creation automatically triggers an initial 'refresh' operation -whereby the programs scans your music files and stores details of every one of them into the database. Only after this 'refresh' does the database accurately record what music you own. Inevitably, therefore, though you get a 'free' refresh as part of the database creation process, you'll want to re-refresh every time your music collection changes. If you acquire new music, or dispose of unwanted works, or correct the tags in this file or re-rip that one... then a database refresh will keep Niente aware of what you've been doing to your music collection. You might consider scheduling (via cron) Niente to run a database refresh every night, for example, so that it captures that day's acquisitions and cataloguing activity. A refresh of the database for my 1.7TB music collection (which stores about 15,000 different recordings of works, and which I consider to be relatively big) only takes a few minutes, so it's not a big deal to refresh quite often.

The real heavy lifting takes place during Niente's performing of an integrity check. During this phase of activity, which is not automatically triggered but always has to be manually invoked, Niente reads its database of what music files exist. It visits each one in turn and captures some of its tag data and, perhaps most importantly, reads the audio signal from beginning to end and calculates a hash value for it -a sort-of checksum that is that music signal's digital 'fingerprint'. It's this bit that takes quite a lot of effort and time: on my 1.7TB music collection, an integrity check of all of it takes around 7 hours to complete.

An integrity check is therefore something you want to happen frequently, so that you can spot physical corruption of your music files as soon as possible and before too much damage has a chance to

propagate too far. It's also something you don't want to perform *too* frequently, though, because it takes ages and gives your CPU and hard disk(s) a bit of a hard time! As we'll go on to discuss, therefore, you'll have to find a balance: a way of running Niente often enough to be useful, but not in a way that your entire music collection is hammered every time!

Anyway: once you've run an integrity check, the results of it are written back to the database.

Only at this point can the useful fourth step of the Niente workflow be performed: reporting on what Niente has discovered. There are various reporting options (as we'll see: **Section 8.3**) and it's entirely up to you how and when you take them. Every time you do, however, Niente simply reads from its own database to determine what records should be displayed on which reports. Niente doesn't need to re-scan anything to produce a report, in short, so report production is always quick and easy.
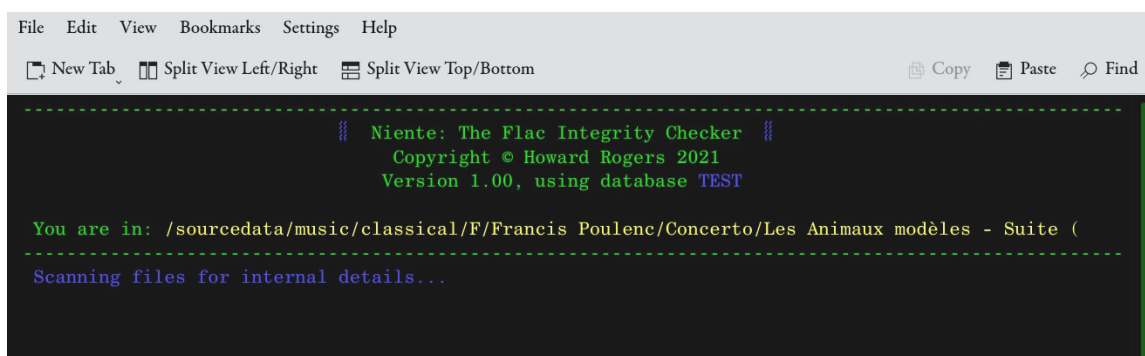
Always bear in mind that you can generate reports at any time -but if you haven't recently refreshed your database, you'll be reporting on a 'set' of FLAC files that once existed in that state, but might not do so any longer. Similarly, if you report without doing a fresh integrity check, the reports produced will tell you how well your music collection *used* to pass its checks. It won't tell you what the current state of your music collection is.

Accordingly, once you've created your database, get into a routine of: **refresh, check, report**. Only in that context will the checks be made on the current state of your music collection, and only then can the reports tells you your current passes and failures.

6.0 The Niente Main Program Display

To be honest, there's not a lot to look at when Niente is running any of the four main steps of its workflow!

Here's what it looks like when it's running an integrity check, for example:

It simply tells you what database it's using (in the third 'title' line, in blue).

It then tells you what folder it's either been invoked from -or, as in this case, what folder it's currently scanning to find music files to check. That's always in yellow, and as it changes, you get a sense of whereabouts in your collection it's got to in its work. In my case, it's got to 'Francis Poulenc', so I know that whilst it's checked all my Benjamin Britten, it hasn't even started on the Johann Sebastian Bach! It's going to be quite a while working, therefore...!

In general, anything you see in green is 'static text': labels, program names, copyright notices and so on.

Anything in blue is 'fairly static text that varies a bit depending on user input'. The database name appears in blue, for example, because it varies, depending on what database you *say* you want to work with. The fact the program is 'scanning for internal details' is also not-quite-static: what the program does depends on what you asked it to do, so it appears in blue.

Anything yellow is quite dynamic text: usually progress indicators of some sort (such as the folder names shown here), constantly changing as the program does it's work.

If you see anything in red, it's an error or a warning of some kind.

7.0 The Niente Database

Niente cannot do anything meaningful without storing data in a database that you get it to create.

- That database has to be created with the `--createdb` run-time parameter.
- It is then kept up-to-date with changes in your music collection with the `--refreshdb` parameter.
- It is thereafter used by Niente any time the program is run with the `--dbname` parameter.

It's important to realise that creating a database doesn't mean moving your physical music files anywhere or altering them in any way. The database is a store of information as to where, on disk, your music files can be found: it does not contain the music files themselves. Accordingly, the database is small and compact, even after a very large music collection has been scanned: my 1.7TB music collection is represented by a Niente database that is only 6MB big, for example.

The Niente database is a sqlite3 one (not MySQL, in case anyone was wondering). Sqlite3 is an open source, compact, highly portable and reasonably-performant relational database.

7.1 Creating a Database

To create a database in the first place, you must run Niente with the following style of command;

```
niente --createdb --dbname=mymusic --musicdir=/path/to/root/of/music-collection
```

The **--createdb** parameter is simply the instruction to create a database: the detail as to what that database's name should be is provided by the **--dbname** parameter, which must therefore be present. The **--musicdir** parameter must also be present, so Niente knows where to find your music collection and thus what the source of its database's records should be.

If you fail to supply a --dbname parameter then you'll receive a warning that a 'no-name' database cannot exist:

```
--------------------------------------------------------------------------------
                    ‖   Niente: The Flac Integrity Checker   ‖
                          Copyright © Howard Rogers 2021
                         Version 1.00, using database XCX

You are in: /home/hjr/Desktop
--------------------------------------------------------------------------------
Warning: You haven't specified a database name, and a database of 'no name' cannot be created!
Please re-run using the --dbname parameter, and supply a usable name when doing so.

Quitting in the meantime.
```

If the database name you supply already exists, you'll again be warned about this and told that no database creation is possible and that you should probably instead be trying to 'refresh' the database which already exists.

If you attempt to create a database without a --musicdir parameter to tell it where the music files it's supposed to check can be found, then yet another warning in red will appear telling you to remember to supply it.

You need all three parameters when creating a database, in short: there are no defaults and without every one of them spelled out, further progress is impossible.

There is no limit to the number of databases you can create, though they must each be uniquely named. If you had classical and jazz music collections, for example, and wished to keep them distinct, you could do two 'createdb' runs, naming each database something like 'classical' and 'jazz', and you'd thereafter be able to scan and check each type of music distinctly with different runs of Niente using different --dbname parameters.

The musicdir parameter should point to the 'root' of your folder tree within which your digital music collection can be found. There is no practical limit to the 'folder depth' that Niente will descend to in order to find FLAC files. In my own case, for example, one of my Beethoven symphonies can be found in the following folder:

/music/classical/L/Ludwig van Beethoven/Symphonic/Jochum (LSO) Cycle/Symphony No. 3 (Jochum - 1976)/

...so you can see that's quite a few folders 'deep': but the root of all my music is simply:

 /music/classical

...and that's the folder I'd add as the value for the --musicdir parameter, and the symphony will be 'discovered' within that path perfectly well.

Should your desired musicdir path contain spaces, please wrap the entire thing in double quotation marks. For example:

```
niente --createdb=music --musicdir="/my music/classical music"
```

Ideally, you wouldn't have spaces in the 'root' of your music folder path like that, but if you insist on it, Niente can at least deal with it, double quotes permitting.

Once Niente creates a database, it sets about populating it by performing an automatic 'refresh database'. What it does during this process is also what happens when you subsequently perform manual refreshes of the database... so let's examine that process now.

7.2 Refreshing a Database

If your music collection is very static, you may never need to refresh the Niente database -but if you acquire new recordings over time, or alter the metadata or physical file/folder names associated with existing ones, then you need to keep the Niente database 'in synch' with the evolving state of your music collection by means of performing a database refresh.

In a database refresh, the contents of the existing main database table are preserved, but new music files are added as new rows to it and any music files which are detected as no longer existing have their corresponding table row(s) deleted. The refresh process is therefore generally quite quick: most existing rows won't need altering at all, and the program simply checks the files they describe still exist. Very little data is collected for any files new to the database, too: the physical folder they're found in is about it. So even the initial scan of a huge music collection won't take too long.

Since a database refresh closely resembles a database (re)creation, the command syntax for performing one is very similar to the way you create a database in the first place:

```
niente --refreshdb --musicdir=/path/to/root/of/music-collection
```

There is no hard-and-fast rule about how often to refresh your database: as mentioned, it really depends on how 'dynamic' your music collection is. If you are acquiring half a dozen new recordings a week, say, then maybe a weekly refresh might be in order; if it's half a dozen a *day*, then maybe nightly refreshes are a better fit. Personally, I schedule a nightly refresh at 2AM in my crontab file with the following entry:

```
# Refresh the main Niente database nightly at 2AM
# ------------------------------------------------------------
0 2 * * * /usr/bin/niente.sh --refreshdb --dbname=main --musicdir=/sourcedata/music/classical
```

Remember to use the full path to the Niente script (i.e., /usr/bin/niente.sh) in crontab, rather than relying on just 'niente', as that requires a PATH variable to work, and PATHs are seldom set correctly in the context of a crontab job.

7.3 Table Structures and Methodology

In normal use, you do not need to know, nor care, about the inner structures of the Niente database - though if you want to construct your own SQL reports and analyses of your music collection, understanding its tables and their contents may be helpful. If you are not skilled in the ways of SQL, however, you can ignore the next few pages: skip to 'Accessing the Database' further on!

The Niente database, when created, contains just two tables, as follows:
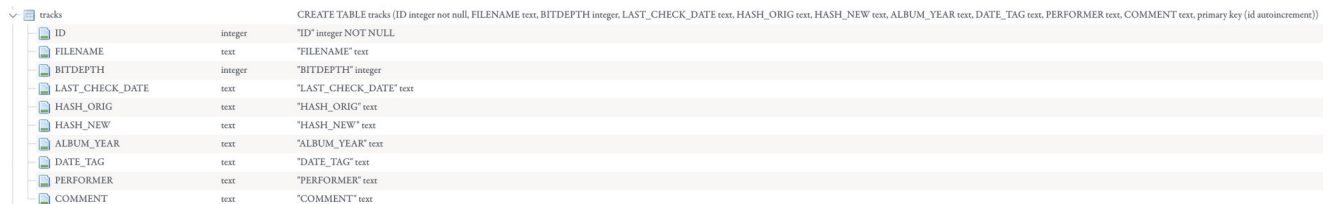
- tracks: populated after the music collection is scanned as part of a createdb or a refreshdb operation. One row represents a single, unique FLAC file, with details of it's location on disk and its file name. Various other attributes (such as whether it's a 16-bit FLAC or a hi-res 24-bit one) are also collected, to help with subsequent integrity checking

- sqlite_sequence: an internal table that records the unique ID numbers given out to tracks when those tables are populated

The TRACKS table has a synthetic primary key (on the ID column) and this column is therefore indexed.

7.4 Schema Description

The following screen captures attempt to show the entire minimalist Niente schema (the database name will depend on what parameter is supplied to the **--createdb=xxxx** parameter).

TRACKS

tracks		CREATE TABLE tracks (ID integer not null, FILENAME text, BITDEPTH integer, LAST_CHECK_DATE text, HASH_ORIG text, HASH_NEW text, ALBUM_YEAR text, DATE_TAG text, PERFORMER text, COMMENT text, primary key (id autoincrement))
ID	integer	"ID" integer NOT NULL
FILENAME	text	"FILENAME" text
BITDEPTH	integer	"BITDEPTH" integer
LAST_CHECK_DATE	text	"LAST_CHECK_DATE" text
HASH_ORIG	text	"HASH_ORIG" text
HASH_NEW	text	"HASH_NEW" text
ALBUM_YEAR	text	"ALBUM_YEAR" text
DATE_TAG	text	"DATE_TAG" text
PERFORMER	text	"PERFORMER" text
COMMENT	text	"COMMENT" text

INDEXES

Indices (1)	
idx_tracks_id	CREATE UNIQUE INDEX idx_tracks_id on tracks(id)
id	"id"

A textual representation of the schema is available from within the sqlite command line shell, which can be accessed from a terminal session with the following commands (which assume the database is named "main", and thus exists as a file called main.db within the standard Niente folder):

```
cd $HOME/.local/share/niente
sqlite3
.open main.db
.tables
.indexes
.schema
.quit
```

(It is the **.schema** command that will display the SQL code needed to re-create the schema completely)

7.5 Accessing the Database

The Niente database is not intended to be directly accessed. The reports that Niente produces from it can be accessed by using the **--report** runtime parameter instead, for example.

However, since the Niente database is a standard Sqlite3 database, it *can* be queried at any time using standard (and free/open source) tools such as the **DB Browser for Sqlite**.

The standard Sqlite3 command line shell can also be used to query the Niente database directly, independently of Niente itself.

Do please note that though the database is configured to run in 'Write Ahead Logging (WAL) mode', such that a reader shouldn't block another reader, nor a writer a reader, if you are accessing the

database at the same time as Niente is trying to record the results of a fresh 'scan' or 'integrity check' in it, unexpected results may occur. It is also generally not a good idea to perform your own inserts, updates or deletes on any part of the Niente database using external tools: corruption to the database may occur if you try.

7.6 Backing up the Database

Fundamentally, the Niente database merely describes your music collection. Lose it, therefore, and you can easily re-create it by scanning your music collection afresh. There is thus no real pressing need to back up Niente's database at all (and I don't bother, personally).

If you wished to do so anyway, however, then a simple copy of the `$HOME/.local/share/niente` folder to a backup destination is sufficient to achieve a functional backup. Restores would simply consist of copying that folder back from the backup device into that specific location on your PC. Note the full-stop ('period') before the '.local' bit of the path: that makes it hidden folder, but once you are inside it, everything just looks and behaves normally.

7.7 Nuking a Database

A new option, introduced in Niente version 2.0 in November 2021, is to be able to 'nuke' a database. You do so by invoking Niente with each of the **—dbname=*xxxx***, the **--musicdir** and **--nuke** run-time option switches (the dbname one is required so we know *which* database to nuke; the musicdir one is required so we know how to re-populate the wiped database once more).

A "nuke" basically wipes a database of all of its data, resetting it into its freshly-created state -and then performing a fresh scan for music files with which to re-populate it from scratch.

Before version 2.0 of Niente, therefore, you could achieve the same sort of thing by simply deleting an existing database (manually, using your file browser, and removing the database stored in `$HOME/.local/share/niente`) and creating a new one, followed by a –refreshdb operation. Having the ––nuke option simply makes it a one-click kind of process that's entirely automated.

After a 'nuke', the Niente database will be empty, which is less than useful! Accordingly, Niente will automatically follow a nuke by a refresh to re-populate the 'cleaned' database once more. A new integrity check is *not* automatically initiated, however: it's time-consuming and CPU-intensive to do one, so Niente leaves it to you to call for that as circumstances allow.

As to *why* you might want to nuke a Niente database? Well, over time, it might get messy: you start with a refresh that populates everything in nicely-alphabetical order, but as you re-catalogue files, remove ones you no longer like and add entirely new recordings to your collection, the records in the database will tend to 'fragment' a little. Things won't be *perfectly* alphabetical any more, for example, because your newly-acquired Aaron Copland recording's data will be stored in the database *after* Zoltan Kodaly's (since newly-identified recordings just get tacked on to the end of the existing records, not slotted in-between). That sort of 'degradation' won't stop Niente working perfectly well, but if it bothers you at all, you can at least now (rarely!) wipe-and-start clean once more.

8.0 Run-Time Parameters

Niente is run, as you've seen, by typing the command `niente` in a terminal session.

It won't actually do anything useful, however, until you *tell* it what to do by passing it 'switches' or 'runtime parameters', which are keywords prefixed with a double-dash (and sometimes suffixed with a value).

There are 12 possible runtime parameters: some you only use once or rarely. Others, you'll use repeatedly.

You can think of them as coming in three 'groups', which I'll call *initialisation*, *routine* and *reporting*.

I'll list each group and discuss the subtleties of each parameter in the following pages.

8.1 Initialization Parameters

These parameters are usually only used once (or infrequently), as part of post-installation configuration or tailoring.

--createdb	Instruction to create a database. Requires the --dbname parameter to specify the name of the database. Also requires --musicdir, so it knows where to scan for music files after database creation.	The database name becomes a physical file name. Spaces should be avoided, unless the entire name is wrapped in double quotes.
--refreshdb	Instruction to rescan a music collection and add to whatever database is specified with the accompanying --dbname parameter. Requires --musicdir.	New or changed files are added; existing and unchanged files are merely checked for continued existence.
--musicdir=*xxxx*	Specifies the physical location of a folder tree structure containing music (FLAC) files. Needs *xxxx* to represent a full path to the root of the collection.	Needs to be present when creating a new database and when refreshing the database and any time a report is produced.
--checkver	Instruction to check to see if a newer version of Niente has been released and to fetch and install it if one is found.	Will prompt for sudo privileges if a new version is found (as sudo is required to copy relevant files to privileged directories).
--licence / --license	Instruction to display the GPL v2 text under which Niente is licensed.	US and British spelling variants both work the same

These 'initialization parameters' tend to be used only once -or, at least, very infrequently. Usually, you'd use them immediately after a fresh installation of Niente, because they setup and configure things which enable much of Niente's functionality. In the sections that follow, I remove the leading '--' before the parameter name for aesthetic reasons: the double-dash is needed when actually using the parameters, though!

8.1.1 Createdb

This parameter triggers the creation of a Sqlite3 database, with a name that matches the value you also supply with the --dbname parameter . You can't use --createdb without *also* mentioning --dbname. The database that gets created is a single file stored in the **$HOME/.local/share/niente** folder with a name identical to the --dbname parameter value. It contains pretty much a single table, which lists the full path and file name of every FLAC file that Niente goes on to discover. See **Section** 7 for more details on this internal database structure.

The **--createdb** parameter always has to be used in combination with the **--musicdir** parameter. That's because immediately after creating the database file, Niente scans the musicdir provided to populate it with data. If it doesn't know where to scan, because you didn't mention a musicdir, database creation will fail, with an appropriate error message displayed. The full database creation command would therefore be something like:

```
niente --createdb --dbname=music --musicdir=/music/classical
```

This will create a database file called **$HOME/.local/share/niente/music.db** and then scan the entire contents of the /music/classical folder to populate that new database.

Note that you can create as many databases with Niente as you like. If you have classical, jazz and 'not very good' music in distinct folder hierarchies, for example, you can easily create three different databases called (perhaps) 'classical', 'jazz' and 'overflow'. You can later choose which database to use for reporting or doing integrity checks by subsequently running Niente with a particular value for the --dbname parameter (as we'll see shortly).

8.1.2. Refreshdb

A static music collection may be scanned just the once, as part of the createdb process, and need never be scanned again: it's static, after all, and thus not changing, so the snapshot of it taken at database creation time is a true picture of that collection.

If , however, you are still acquiring new music to add to your collection, this one-off 'snapshot scan' of a folder structure at database creation time will not meet your needs: if the collection is changing, the database contents need to be periodically refreshed to make it keep up with those changes, which is why you'd use the **--refresh** database parameter in a command such as:

```
niente --refreshdb --dbname=music --musicdir=/music/classical
```

This triggers Niente to take a fresh look at the contents of the folders contained within the folder again specified by the --musicdir parameter value. Note we need to know *which* database to refresh: the --dbname parameter is therefore required to provide this vital information.

The existing contents of the database are not wiped by a refresh operation. However, every file already listed in the database will be checked for its continued existence: if Niente can't find it in the place it was previously said to exist, it will delete that row from the database. On the other hand, if it comes across a new FLAC file as it scans the --musicdir folder structure, it will *add* the details of that file into the database.

This means that the list of files in the database won't always be ordered 'correctly': everything is initially loaded in A→Z order, but if you added a new Britten opera to your music collection and then did a --refreshdb, you'd find the new Britten tracks get inserted into the database table at the end ...so they would come after the 'Z' entries. This isn't a problem, however: you need never know or worry about the order of rows inside the database table: Niente handles all that for you.

A refresh of the database is a relatively 'light' operation: it involves some disk activity as the musicdir is scanned, but there is no deep reading of the audio signal of any file found, nor is any complicated mathematical calculation to generate digital 'fingerprints' of files undertaken. Accordingly, you can expect a 2TB music collection to be re-scanned in maybe less than half-an-hour, given a CPU manufactured in the last 15 years. Your PC's RAM will not be overwhelmed, either: you will barely notice any impact on memory consumption as the scan proceeds.

Accordingly, if you are regularly adding to your music collection, or correcting its file names or tag entries, you may want to schedule a routine refresh via your crontab. Here's my crontab entry, for example:

```
# Refresh the main Niente database nightly at 4AM every day bar Sunday
# ------------------------------------------------------------------
0 3 * * MON-SAT /usr/bin/niente.sh --refreshdb --dbname=main --musicdir=/music/classical
```

Remember that in crontab, one tends to specify the full path to executables, so here it's /usr/bin/niente.sh, not just 'niente'. You can see that I refresh my database nightly at 3AM, when no-one's around to see it happen, every day of the week except Sunday. I skip the refresh on Sundays, because Sundays are my days for doing integrity checks (see **Section 8.2.4** below).

8.1.3 Musicdir

As already mentioned, this parameter takes a value of the full path to the root of a folder tree containing a set of FLAC files. It is required for database creation and refreshes.

In case it's not obvious, you can only specify *one* music folder when creating or refreshing a database, so the path supplied must be 'high up enough' to allow the collection scan to capture all appropriate music in one pass.

If you had a folder structure like this, for example:

```
/Benjamin Britten
/Classical Music/Beethoven
/Gustav Mahler
```

...it would be difficult to have all three composers included in one database unless you said your musicdir was '/' (the root folder)... and a scan of your system's root folder is likely to be long and painful!

Better would be something like this:

```
/music/Benjamin Britten
/music/Classical Music/Beethoven
/music/Gustav Mahler
```

...for now you can scan '/music' and get all three composers collected in one hit. It doesn't matter for these purposes that Beethoven is one folder further 'down' than the other two: Niente will find music within a top-level folder no matter how far down the 'tree' it might be. There *are* aesthetic reasons for wanting a consistent folder hierarchy and structure, but core FLAC file discovery is not affected by such questions of aesthetics!

8.1.4 Checkver

The --checkver parameter is something you will want to run infrequently, but regularly (say, once a week or once a month). The command:

```
 niente --checkver
```

...simply forces Niente to check the Internet to see if a more recent version of the program available. If there is, Niente will offer to fetch and install it for you. If there isn't, it will simply say 'you're already running the latest version'.

Should a more recent version be discovered, you will be asked whether you want it fetched and installed. If you agree to do so, you will be prompted to supply your sudo password (because installing Niente involves writing into the /usr/bin folder, which ordinary users cannot do, but which root can). The upgrade will always overwrite the *program* files involved, but will never alter in any way the contents of your music database. Niente upgrades are therefore quite safe. It is important to perform this version check reasonably often, because new software releases fixing bugs and adding new functionality are being continuously made, with no fixed schedule.

8.1.5 License/Licence

This parameter can be spelled in either way, depending on whether you prefer American or British English spelling! In either case, if you run Niente with this parameter present, the full text of the GPL v. 2 license is downloaded and displayed within the **less** program (meaning that you can up- and down-arrow or PgUp/PgDn to scroll your way through it in either direction). Press 'q' to quit the display and exit back to the command line. The parameter has no real utility, other than to conveniently display the full text of the license under which Niente is made available. It's not an option you're going to use very often, I think!

8.2 Routine Parameters

These parameters tend to affect the way the program looks or behaves and are accordingly parameters you will want to specify frequently (or at least, some of them!).

--displaycolour=*xxxx* --displaycolor=*xxxx*	Valid values are light, dark, neutral, standard and classic	Alters the colour display of the program text. The default is 'standard'
--dbname=*xxxx*	The database name you want to create, refresh, use as a source of file information for an integrity check, or report from.	Must be present when various other parameters are used, so we know what database to use to store data in or read data from.
--checkdays=*y*	The number of days old a record has to be to be a candidate for normal integrity re-checking	The default is 27. Non-numeric values of *y*, or values less than 1, are converted to the default 27 days. Files checked within this number of days won't ordinarily be selected for a re-check.
--integritycheck	An instruction to perform a detailed integrity check on candidate FLAC files referenced by the database.	An integrity check performs a detailed analysis of the audio signal within a FLAC file and creates a digital 'fingerprint' for the file (which hopefully matches the one created when the file was first created).
--force	An override for the --checkdays parameter: forces *all* database records to be subject to an integrity check, regardless of when the last one was performed.	
--nuke	Wipes the database and returns it to its freshly-created state (i.e., with no records at all). Should be followed with a --refreshdb and a new --integritycheck to re-populate	Requires --dbname, so we know which database to wipe, and --musicdir so we know how to re-populate it.

8.2.1 Displaycolour/Displaycolor

However it's spelled, this parameter takes one of four possible values, with 'standard' being the default. Any value passed that isn't 'light', 'dark' or 'neutral' is silently interpreted as 'standard'.

The standard colour scheme involves the use of cyan, green, red and yellow text: users with colour perception issues may prefer trying one of the other three values, which essentially turn the display into monochrome white ("light"), monochrome black ("dark") or monochrome yellow ("neutral").

Version 2.0 of Niente changed the standard colour scheme from one using a dark blue to one using a much lighter blue (technically, 'cyan'!) for legibility reasons. Anyone wanting version 2.0 to revert to the use of dark blue can do so by asking to use the **classic** displaycolour scheme.

Note that the various colour schemes might not always display correctly unless you *also* configure the appearance of your terminal session. If you ask Niente to display 'dark', for example, when your terminal profile already mandates a dark appearance, you'll be basically looking at black text on a black background!

Here are screenshots of the various colour options available to you via the use of this parameter.

First, 'standard', which is the default colour scheme used in Version 2.0 of Niente:

Broadly speaking, the standard display uses Green to display static program text; Yellow to display dynamic text whose content will depend (maybe indirectly) on something the user supplied; Red to display error messages or alert condition warnings; and light Blue to display static text that depends on user input (such as the name of the database in use).

If you prefer the colour scheme previously used as the default in Version 1.0, you can still use it, but only by asking to use the **classic** colour scheme, which then looks like this:

It's practically identical to the new Version 2.0 standard colour scheme, except that a dark blue is used where the newer version uses cyan (or 'light blue'). The 'meaning' of each colour (i.e., whether it's used to indicate user input, a warning, or merely as static display text and so on) remains the same in either case.

Next, here is what the Light colour scheme looks like:

```
------------------------------------------------------------------------------
                        || Niente: The Flac Integrity Checker ||
                             Copyright © Howard Rogers 2021
                           Version 2.00, using database MAIN

 Current Operation: Loading database with details of discovered music files...
------------------------------------------------------------------------------

 Processing: …cal/G/Georges Bizet/Opera/Carmen (Abbado - 1977)/Carmen (Abbado - 1977)-24-192000.flac
 (Recording 6512 of 17260)
```

Everything becomes white monochrome text, basically, so that there's no difference between program text, user information and error messages.

Now the "Dark" theme:

```
------------------------------------------------------------------------------
                        || Niente: The Flac Integrity Checker ||
                             Copyright © Howard Rogers 2021
                           Version 2.00, using database MAIN

 Current Operation: Looking for music files...
------------------------------------------------------------------------------

 Searching in: /sourcedata/music/classical (this may take a while...)    /
```

Here, I've altered the terminal theme to be 'Tango Light'… and now you can see that the dark Niente theme results in monochrome black text on a now-white background. If you didn't alter the terminal theme from its usual default, of course, you'd end up with black text on a black background… which would be somewhat less than useful!

Finally, here's Neutral:

```
------------------------------------------------------------------------------
                        || Niente: The Flac Integrity Checker ||
                             Copyright © Howard Rogers 2021
                           Version 2.00, using database MAIN

 Current Operation: Loading database with details of discovered music files...
------------------------------------------------------------------------------

 Processing: … 1068 (Müller-Brühl - 1998)/Orchestral Suite No. 3 BWV 1068 (Müller-Brühl - 1998).flac
 (Recording 10709 of 17260)
```

Again, I've had to adjust the terminal theme's background colour as well as specifying a new value for the –displaycolour parameter, otherwise you'll end up with bright yellow text on (potentially) a very bright background, which is practically unusable. With a suitable background colour, however, the neutral theme will give you, essentially, a monochrome yellow text display -the idea being that yellow-on-dark is less tiring on the eyes than plain black-on-white or white-on-black.

8.2.2 Dbname

Almost any time you run Niente, you'll have to mention the --dbname parameter. It tells Niente where the 'source of information' about the FLAC files it's supposed to check can be found. It also tells Niente

where to store the results of its scans of your FLAC files. Without it, Niente hasn't got a clue where to start scanning nor where to write the results.

Specifically, if you say **--createdb**, **--refreshdb** or **--report**, you *must* specify --dbname and the program will error out with a suitable warning if you don't.

Since the database name becomes a physical file name, it's a good idea to make your database name a simple, short, one-word value without spaces. Good names might be, for example, 'music' or 'main' or 'overflow'. A bad name might be something like 'My main database', because all those spaces might cause trouble further down the line. If you insist on using spaces within a database name, you'll have to wrap the entire parameter within double quotation marks, so you'd have to end up saying something like **--createdb="My Main Music Database"**.

I would recommend not using capital letters either: Linux filesystems are case-sensitive, so if you mix cases in the database names, you'll always have to remember the precise combination of upper- and lower-case letters for all future uses. Sticking to one-word, lower-case database names makes life easier all round!

For the same 'it becomes a file name' reason, avoid characters which your file system would reject as illegal or which have significance for the file system: so avoid asterisks and forward slashes, to name but two!

8.2.3 Checkdays

When you perform an integrity check (see below, Section 8.2.4), Niente's standard operating procedure is to read from its database and find files there which have a 'last checked date' that is at least 27 **days** before 'now'. It takes the view that there is not a lot of point in doing heavy-duty integrity checks of files which were only checked a day or two ago, for example.

The --checkdays parameter must be given a numeric value, greater than 1. There is no upper limit, so --checkdays=9999 would be valid, though pretty silly!

If you supply an illegal value for the parameter (such as --checkdays=-20), the default value of '27 days' kicks in silently (negative numbers are not allowed, in other words).

8.2.4 Integritycheck

This parameter is really the reason Niente exists in the first place! It's an instruction to perform a detailed integrity check on the FLAC files listed in its tables. That integrity check is quite CPU and disk-activity intensive, so it's not something you necessarily want to be doing to your large collection of FLAC files every few hours! It is, however, completely non-destructive: it doesn't alter the content of your music files *at all,* though it does make the files read-only as part of its processing (on the grounds that a file which cannot be ordinarily written to is less likely to be modified in a way that corrupts its audio signal over time).

Note that this parameter **requires** --dbname, but definitely does not need --musicdir: Niente doesn't need to be told where to scan for FLAC files, since its database tells it precisely where every FLAC it's ever seen during a database refresh lives on disk.

An integrity check of your entire music collection can take quite a while to complete: my 1.7TB collection takes about 6 or 7 hours to process completely. My own crontab entry for a check of the entire collection looks as follows:

```
# Weekly integrity checks of the music source, every Sunday at 1AM
# ------------------------------------------------------------
0 1 * * SUN /usr/bin/niente.sh --integritycheck --dbname=main
```

The computer starts early, at 1AM on every Sunday, so that if the job required re-checking every file, it would be mostly over by the time I struggle out of bed at around 8AM!

But, of course, this *won't* check everything <u>every</u> Sunday. On most Sundays, the last check date for most files will be within 27 days ...and, as you'll notice, I'm not setting --checkdays in my command, so the default of 27 will be used. Only after 4 weeks (i.e., 28 days), therefore, will most of my files fall into the 'checkdays net' and <u>actually</u> be re-checked.

The good news is that if I acquire new CDs throughout the month (as I do), then this 'run every Sunday but skip most things most times' arrangement will mean that something I catalogue and add to the collection on, say, Thursday will get integrity-checked just three days later. Worst case, if I add something to my music collection on Monday, only 6 days will elapse before it gets picked up for an integrity check.

So, it's fine to schedule the integrity check to happen quite frequently -because, unless you say --force (see below, **Section 8.2.5**), or otherwise use a very low --checkdays, hardly anything except new acquisitions will *actually* get checked each time the program is run. Only every 27 days (or whatever other --checkdays you supply) will the nightly-scheduled job suddenly have to deal real work on large proportions of your music collection.

The results of an integrity checked are written back to the database mentioned in the --dbname parameter (which must therefore be present). Specifically, the 'LAST_CHECK_DATE' column is updated with the current date, and the HASH_NEW column is updated with the hash value freshly computed by re-reading the audio signal in every file checked. It is by comparing the contents of the HASH_NEW column thus freshly computed with the contents of the HASH_ORIG column (which contains the hash value originally stored in the file at the time of its creation) that we are able to determine whether any FLAC file has become physically corrupted over time.

8.2.5 Force

There may be occasions when you do not want to wait for the routine integrity check scheduled in your crontab to take place: you need a check to be performed *now.* You could achieve that by running Niente with **--integritycheck --checkdays=1**, since that would mean 'please check everything not previously checked within the last day' (which is probably going to be all your FLACs).

However: that's not quite exact! The LAST_CHECK_DATE is always updated to the *start* of a day (i.e., you run an integrity check at 6pm on 28th June 2021: the LAST_CHECK_DATE will say the check took place at midnight on that day (i.e., as the 27th became the 28th). That rounding is usually fine: we tend not to worry about hours and minutes in these sorts of calculations! But it *might* mean that --

checkdays=1 doesn't guarantee a re-check of everything, depending on when it's issued and when you ran your last integrity check.

The --force parameter removes any such ambiguity: it forces Niente to do an integrity check for every FLAC file it knows about with absolutely no regard to when it was previously checked. It will, 100% guaranteed, cause every file for which it stores details to be re-checked.

8.2.5 Nuke

Over time, the internal contents of the Niente database may get disordered (as you delete old recordings, re-catalogue existing ones and acquire new ones, for example). This doesn't stop Niente working -but, if you're accessing the database with third-party tools, it can be disconcerting to see. A "nuke" therefore deletes all data from Niente's database and then re-populates it by performing a fresh scan for music files. Accordingly, the --nuke option must be specified along with the --dbname and --musicdir parameters. It's not an option you'll take very often, but it's there if you feel you'd like to start fresh, without creating an entirely new database from scratch.

8.3 Reporting Parameters

The Niente database will store the facts of whether your music collection contains physical or logical corruption -but if you don't ask it to report on its findings, you'll never know about them! The reporting parameters are how you get access to that vital information:

--report=*xxxx*	An instruction to produce a report of the plays previously recorded within a database. Valid values for *xxxx* are either a full path and filename for a written report, or the word 'screen' to display the report on-screen.	Requires the --dbname parameter, so we know what database to report on.
--report-type=*xxxx*	The specific integrity report or reports to produce. Valid values for *xxxx* are 'integrity', 'dates', 'performers' or 'all'.	The default is to only produce the 'integrity' report on physical corruption.

8.3.1 Report

The --report parameter takes two possible values: a full path and filename if you want the report that's about to be generated written to a text file; or the word "screen" if you want the report displayed within the terminal session.

Note that the **--dbname** parameter must be present whenever --report is mentioned, because without it, we cannot know what database you want to query to generate the report in the first place. That database needs to be the same one you scanned when creating or refreshing the database, and the one you use when telling Niente to perform integrity checks.

8.3.2 Report-type

Niente can produce three types of report: the physical integrity report, indicating files where the freshly-computed hash value doesn't match the one stored in the file at the moment of its creation; and the two reports on logical inconsistencies between the year stored in the ALBUM and DATE tags, and the inconsistencies that might exist between the contents of the PERFORMER and COMMENT tags.

Accordingly, the --report-type parameter takes the values 'integrity', 'dates' or 'performers'.

A fourth value is possible should you want all three reports in one hit: 'all'.

Depending on whether you said --report=screen or --report=/path/and/filename.txt, the contents of the reports will either appear in your current terminal session or as plain text files in the location specified.

If the report is directed to screen, the lines of the report will be scrollable up or down one line at a time as the mood takes you, using your Up and Down arrow keys. You can also scroll through multiple lines in either direction more quickly using your PgUp and PgDn keys. When you are finished reading the report on-screen, just tap the letter 'q' to quit.

If you have selected the 'all' report type, each of the three reports will be displayed on-screen in turn. You tap q to quit out of one and the second will be displayed automatically. Tap the letter q again to quit out of this second report and the third will automatically display. Only on the third tap of the letter q will *that* report quit: now you'll be returned to the command line from where you requested the reports in the first place.

If you selected a path and filename as the --report parameter value, then text files of the reports will be written out to your chosen destination, but the word 'integriy', 'dates' and 'performers' will be inserted into your specified choice of filename. That is, if you had said :

```
niente --report=/home/reports.txt --report-type=dates --dbname=main
```

...then you'd actually get a text file called /home/reports-dates.txt output. If you selected a report type of 'all', then all three files are output at the same time, each with the '-integrity', '-dates' and '-performers' keyword insertted into their names before the .txt extension.

The reports simply list the files that have failed the various checks, and each type of report performs two separate checks, as follows:

1) **Integrity**: checks that (a) all files have been integrity checked and (b) that all files have freshly computed hash-values which match their original values (indicating no internal corruption).

The 1(a) test is there to deal with the situation where you refresh your database (which might add new rows into the database) but then go straight to doing a report, without remembering to first do an integrity check. The new rows added by the refresh will be listed on the report as awaiting their first integiry check. It's a reminder that the work flow should always really be refresh-check-report.

Obviously the 1(b) test is the one that 'really counts'! It's the one that spots any rows where the original hash value for the audio stream no longer matches the freshly-computed value. That sort of change can only come about by the audio stream data itself being modified from the original -even if only by the

flip of a single bit. That's the fingerprint of internal corruption and can only be fixed by restoring the file from a backup, or re-ripping the music from the original disk.

2) **Dates:** checks that (a) all files have a DATE tag value of some kind and (b) that the DATE tag value matches the year element found in the ALBUM tag, assuming that your 'extended album names' are written in the form of 'actual composition name (performer - recording-year)'. The recording year extracted from the last few characters of the ALBUM tag should logically match the YEAR tag entry, and this test will tell you if they don't for some reason.

3) **Performers:** checks that (a) all files have a PERFORMER tag of some sort; and (b) that it matches some part of the COMMENT tag. The assumption is that if you say a recording is performed by 'Herbert von Karajan, Berlin Philharmonic Orchestra' in the COMMENT tag, the PERFORMER tag will say 'Herbert von Karajan'. If you'd said 'Herbert von Karajan' in the one and only 'Herbert Karajan' in the other, the missing 'von' would trigger a report on the discrepancy.

If you do not use PERFORMER, ALBUM, DATE and COMMENT tags the way Niente expects you to, those last two reports need never be produced and their contents needn't bother you. However you tag, though, the integrity report is important as it detects physical changes to the internal data making up your audio stream, independent of what the tags in that file might or might not be. Data integrity is something everyone needs to worry about -and is at the core of why Niente was written in the first place.

There is always scope for adding more logical checks to Niente in the future, though they will again depend on you adopting a particular tagging strategy. No additional logical checks are currently planned, however.

9.0 Example Reports

I thought I should conclude this manual with some examples of reports produced on my own, real, music collection -and how I responded to them to make things go away. Sometimes it's pretty obvious what 'the fix' should be. Sometimes, less so!

9.1 A Date Mismatch
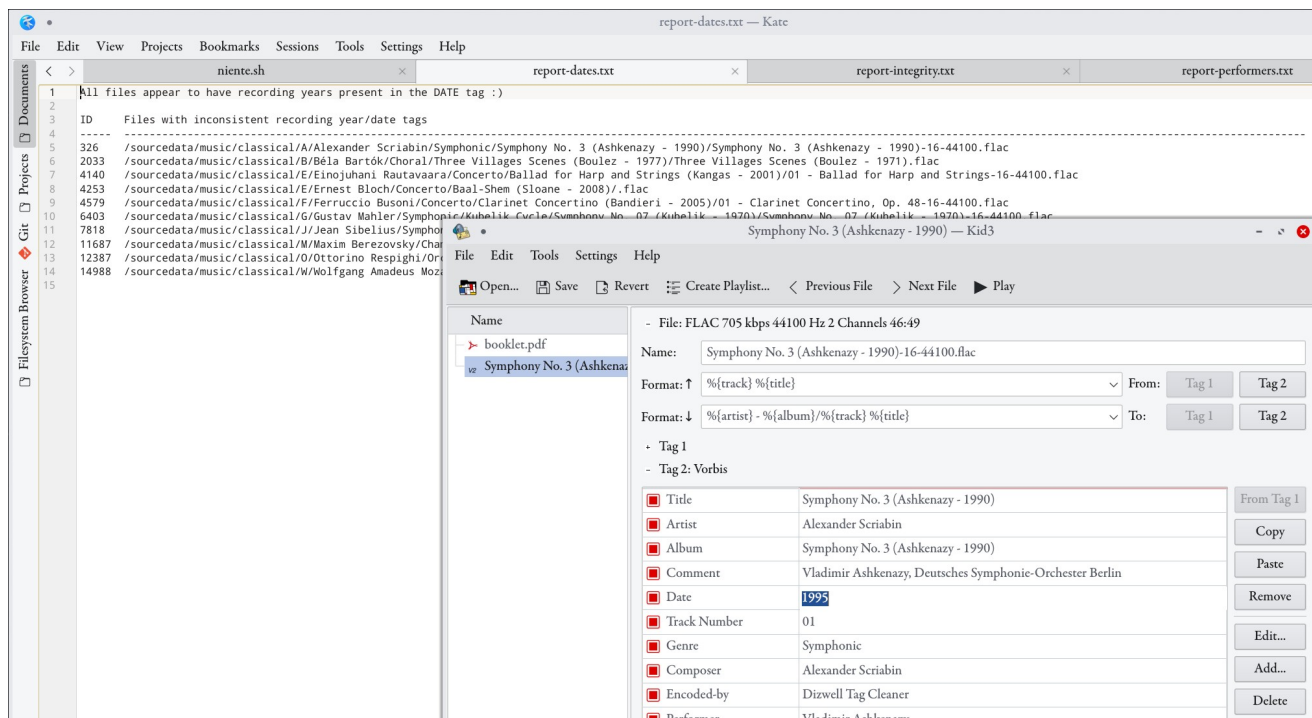
In this screenshot, you can see a report on Date Consistency in the background. Notice that all my music at least has *a* date tag, which is something! Ten files, however, have date tags which are inconsistent with the year recorded in the album tag. I've opened up the first one listed (the Scriabin) in the **Kid3** tag editor in the foreground (any tag editor would do, but this shows things nice and clearly).

You can see that the DATE tag says '1995', but the ALBUM tag says the recording was conducted by 'Ashkenazy - 1990': the fact that 1990 isn't the same as 1995 is why this line was included on the report.

What was the fix in this case? Well, I simply had to make the two (actually three, since the year also happens on this occasion to be listed in the TITLE track too, though Niente wouldn't have spotted that: it *only* compares DATE to ALBUM) years agree.

But should the DATE become 1990? Or should the ALBUM become 1995? There's no easy way to answer that, which is why Niente cannot fix these issues for you! In my case, it was time to dig out the original booklet and check the recording details:

Producers: Andrew Cornall (CD 1), Michael Haas (CDs 2 & 3)
Recording engineers: Stanley Goodall, Michael Mailes (CD 1), Simon Eadon (CD 2), John Pellowe (Rêverie, Symphony No.3), Stanley Goodall (Le Poème de l'extase)
Recording locations: Jesus-Christus-Kirche, Berlin-Dahlem, May 1990 (Rêverie, Symphony No.3), June 1990 (Le Poème de l'extase), September 1995 (CD 2); Schauspielhaus, Berlin, June 1994 (CD 1)
Mastered by Emil Berliner Studios
℗ 1991 (CD 3), 1995 (CD 1), 1996 (CD 2) Decca Music Group Limited
© 2003 Decca Music Group Limited
Introductory notes and translations © Decca Music Group Limited
Booklet editor: Fabian Watkinson for White Label Productions Limited
Cover photo: Photodisc
Booklet cover: Alexander Scriabin, unfinished sketch. Photo: Decca
Art direction: White Label Productions Limited

A close reading shows that Symphony No. 3 was recorded in May 1990... so the DATE is wrong and ALBUM right. I therefore altered the DATE tag on the original file.

Applying 'the fix' does NOT make the line disappear from the report, if I happened to ask for it to be produced again. Changing your music collection does not result in an automatic change to the Niente database. So, having updated the tags, I'd need to do a --refreshdb and a new --integritycheck to get the contents of the report to update to reflect the new state of my music collection.

9.2 A Misleading Date Report

I thought I should show you another example of how you might think about fixing up something listed in a report. Here's another example from the Date report you've just looked at:
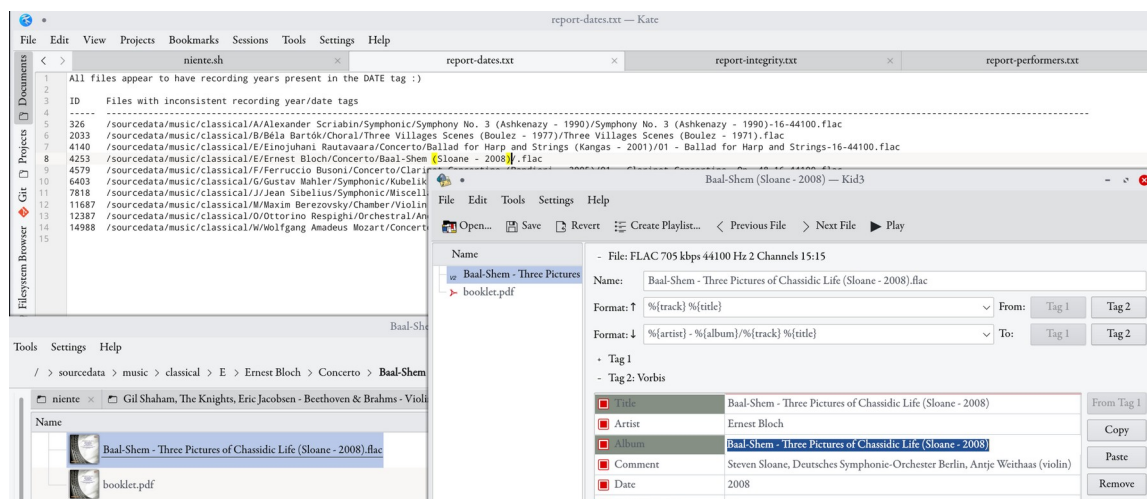
This is from the fourth line in the 'inconsistent dates' report: Ernest Bloch's *Baal-Shem*. As before, I've opened up the file indicated with Kid3... and am puzzled to discover that the DATE is 2008...and so is the ALBUM year element. There's no inconsistency there, so why is it being reported on as though there was?

Well... take a look at the fourth line in the report at the background again. It's listing **Ernest Block/Concerto/Baal-Shem (Sloane - 2008)/.flac.**

I spotted straight away that the file name part of that is actually missing: it just says '.flac', whereas you can see in the lower left the file name is fully-formed as **Baal-Shem - Three Pictures of Chassidic Life (Sloane - 2008).flac.**
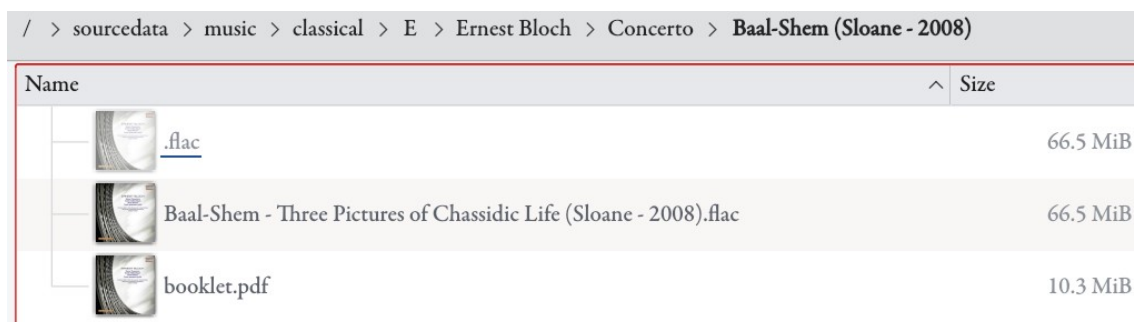
So my first thought was that Niente had been fooled into not reading the file name correctly for some reason -and I noticed that in this file name there are *two* occasions where the sequence 'space dash space' appears: ...Shem - Three... and ...(Sloane - 2008)... I figured that maybe the first space/dash/space had fooled it into thinking it was the date space/dash/space, that it had therefore triggered a mis-read, and therefore, I should replace that first 'space/dash/space' with something like 'space/en-dash/space'. The en-dash is a slightly longer dash than a vanilla hyphen and uses a different Unicode number, so it would look similar to a hyphen, but would not be programmatically read as one.

So I changed the physical file name to **Baal-Shem – Three Pictures of Chassidic Life (Sloane - 2008).flac** ...and if you look carefully, the hyphen between 'Shem' and 'Three' is now a bit bigger than it was before.

After that, I refreshed my database, re-checked integrity for everything (which took 6+ hours!) and... the exact same line on the report still appeared!

Well, I went back and re-named and re-tagged the file again, religiously avoiding any hyphens and converting everything to en-dashes, refreshed the database, re-checked integrity again... and *still* the same line in the report persisted!

You know what it was that was actually causing the report 'error'? Take a look:

/ > sourcedata > music > classical > E > Ernest Bloch > Concerto > **Baal-Shem (Sloane - 2008)**	
Name	∧ Size
.flac	66.5 MiB
Baal-Shem - Three Pictures of Chassidic Life (Sloane - 2008).flac	66.5 MiB
booklet.pdf	10.3 MiB

Once I turned on 'show hidden files' in my Dolphin file manager, I could see that there was actually a copy of the actual music file called, simply, '.flac'... and the leading dot in its name makes it a hidden file, which is why it appears in faint grey tones in this screen capture.

I have no idea how that happened, but in re-naming the file at some point in the past, I must have accidentally created a copy of it without any names -and without any tags either. So, the fully-named file you see listed was perfectly OK and Niente was not being fooled by anything. The report was actually 100% correct, but I just mis-read or mis-interpreted: it was saying there's a file called '.flac' (and there was) and it had inconsistent tags (and it did!). I just couldn't spot or fix the problem because (a) I don't normally have 'show hidden files' switched on, and (b) I'd assumed that Niente was getting the file name I could see wrong for weird reasons (when it wasn't!)

The fix in this case was trivially easy: delete the hidden file. Refresh once more; perform yet another integrity check... problem solved!

9.3 A Performer Problem

Here's an example of a problem detected by the Performer consistency report:
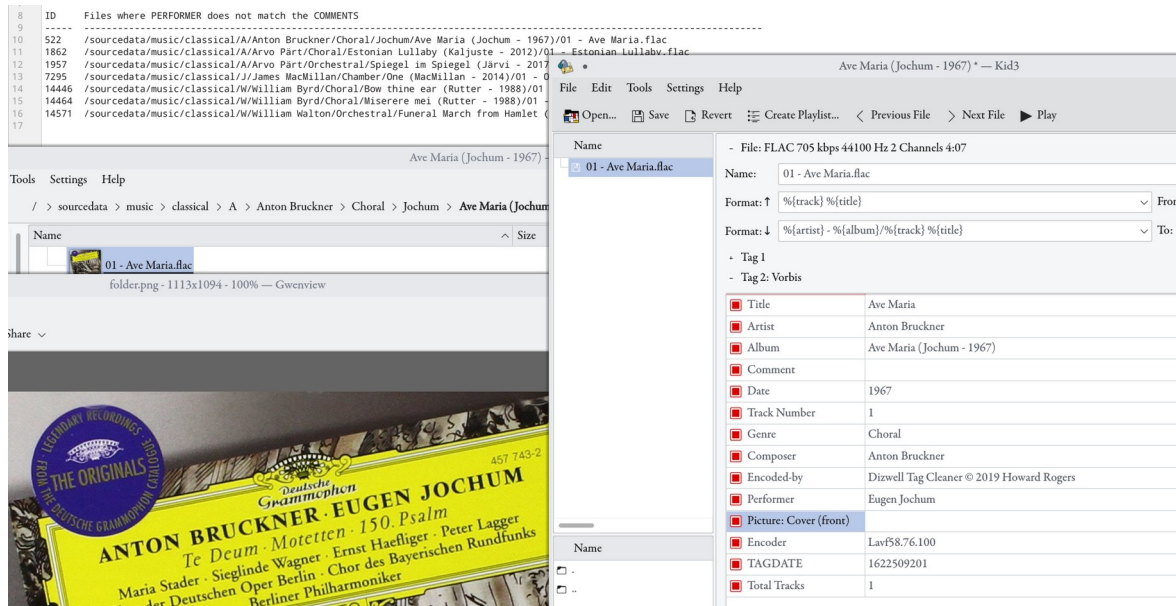
The first line of the text report shown in the background here says what the problem is: the PERFORMER doesn't match the COMMENTS in a recording of Bruckner's *Ave Maria* conducted by Eugen Jochum.

As ever, I open the file in the Kid3 tag editor and, sure enough, there is indeed a mismatch between the two tags... but not in the way you might expect! Eugen Jochum is mentioned perfectly correctly in the PERFORMER tag, so that's not an issue. Unfortunately, for some reason, I seem to have completely forgotten to enter any data for the COMMENT tag at all.

It's not that I spelled the names incorrectly or anything, in other words. The problem is that I just didn't spell anything at all in the COMMENT field!

The fix here was fairly easy, because all the information I needed to fill in the comments was visible in the album art embedded in the music file: Eugen Jochum, Berliner Philharmoniker. Add that into the COMMENT tag, save the file... refresh the database .... re-perform an integrity check... that line disappeared off my next report, no trouble at all.

9.4 A Second Performer Problem

The second line of that previous Performer report mentioned an issue with my recording of Arvo Pärt's *Estonian Lullaby*. See if you can spot the issue:
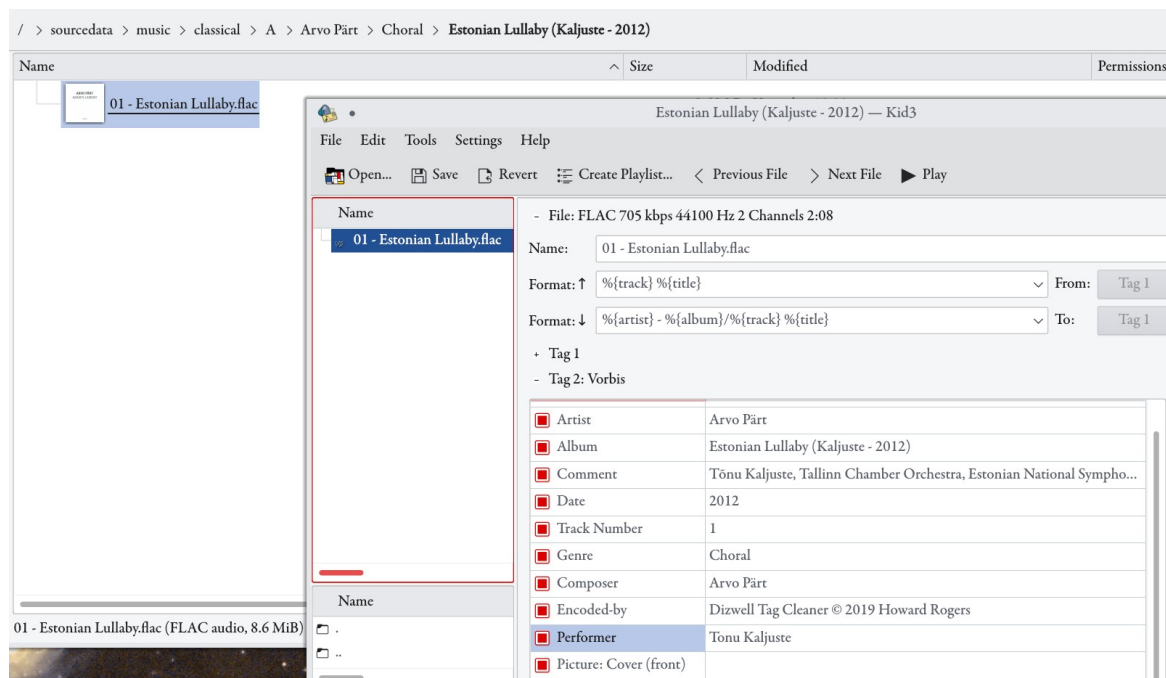
This one's not really difficult, conceptually... but it took me a good couple of minutes to spot it! The COMMENT mentions Tonu Kaljuste, Tallinin Chamber Orchestra and so on. The PERFORMER tag mentions Tony Kaljuste... so why exactly is this being reported on as an inconsistency?!

Memo to self: always check your diacritics!

The COMMENT does *not* menion 'Tonu Kaljuste' but calls him Tõnu Kaljuste. The first o isn't a plain English 'o', but the Unicode character 'small letter o with tilde' ([it's character_00F5, if you care](#)!) which happens to be the 27th character of the Estonian alphabet... and since Mr. Kaljuste is Estonian, it's entirely appropriate to spell his name as he would do.

So I got my COMMENT tag entirely correct... but seem to have slipped into monoglot English when typing in the PERFORMER tag, where he really did get entered as 'Tonu'. I suppose I'm grateful I didn't call him 'Tony'!

Anyway, the fix is simple: just type the correct Unicode character into the PERFORMER tag so it really does match the version of his name in the COMMENT one....and then, as ever, re-scan the database and perform a fresh integrity check. Problem solved.

I've shown these real-life examples of what can happen to your tags to make them internally inconsistent (sloppy typing, inability to spot a diacritic when needed, simple oversight and, finally, a very odd problem with the inadvertent creation of a hidden file on disk!) to show you the kind of curatorial activities Niente's reports can prompt you into doing. The result is a tidier, more coherent and consistent music collection, at the cost of some minor bits of housekeeping!

Appendix A:
Differences between Niente and AFC

If you were at all familiar with running the Absolutely Baching Flac Checker (AFC), which did a similar sort of job to Niente, you might wonder in which ways the two programs differ from each other. Here is a list of the key differences:

- Niente uses a relational database in which to store the results of its integrity tests and to determine the files it should check. AFC simply searched for FLAC files direct off hard disk and stored the results of its integrity checks within the file being checked.

- Niente never modifies the data content of the files it checks; AFC did. Accordingly, AFC triggered cascading backups: since the source file had been changed, its backup needed to be replaced by a fresh copy from source. With AFC checking a 1.7TB music collection once a month, there would need to be 1.7TB of 'new' data pushed to all backup destinations once a month -including to cloud-based ones. Since Niente never modifies the files it checks, Niente causes no such 'backup cascade'.

- Both AFC and Niente perform a fresh calculation of the MD5 checksum of the audio stream of a FLAC file in exactly the same way. They do it using ffmpeg with 16-bit or 24-bit calculations, depending on the perceived bit-depth of the original audio stream. The ffmpeg-calculated hash is then compared to the hash extracted from the FLAC itself via metaflac (which is invariant since the time of the file's creation).

- Niente parallelises the initial search for FLACs when a database is first created. AFC never parallelised any part of its operation. Niente's degree of parallelism is always the number of threads available on your CPU (so a 4-core, hyper-threaded CPU will do the initial FLAC search with a degree of parallelism 8).

- AFC produces textual log files after each run, which you had to visually inspect to see whether or not files had failed their checks. Niente never produces a log file: its 'log' is the rows stored in the database. You 'inspect' those rows at any time by running Niente with the various --report runtime parameters.

Appendix B:
The Requirement for Sudo Privileges

Niente does **not** require sudo (i.e., acquired root privileges) for ordinary operation, but you will need it for installation and performing software updates.

Sudo privileges mean you stay you, and you supply your password, but you acquire temporary rights to run with root privilege and are thus allowed to create or modify files in restricted folders (like /usr/bin). It's a safer way of doing things than *becoming* root directly, which would give you permission to do anything at all for as long as you liked.

Most distros, during installation, tend to ask you for a user account and *automatically* give that initial user account sudo privileges, so you do not then need to do anything at all. Niente or related scripts will prompt you for your 'sudo password' if necessary, and you just supply your ordinary user account password at that point, and all will be well.

But not all distros do this: Debian, for example, does not and trying to run a sudo command produces this foreboding error:

...which is quite funny on a home PC where the only person to report to is yourself!

Anyway, for distros where you cannot automatically exercise sudo rights, here's the workaround. First, become root. Second, set the default editor to **nano** or even a graphical text editor of your choice: if you need help setting sudo, you do not want the pain and hassle involved with using most distros' default text editor, which is vi. Finally, you invoke the **visudo** program and add one line of text to the existing file and save it.

It's just four steps, in other words. Here are the first three, spelled out, as you'd type them in a terminal:

```
su - root
export EDITOR=nano
visudo
```

The first command requires you to be able to provide *root's* password when asked. Assuming you know that, you're in business. When the visudo program runs, you want to find the line which reads:
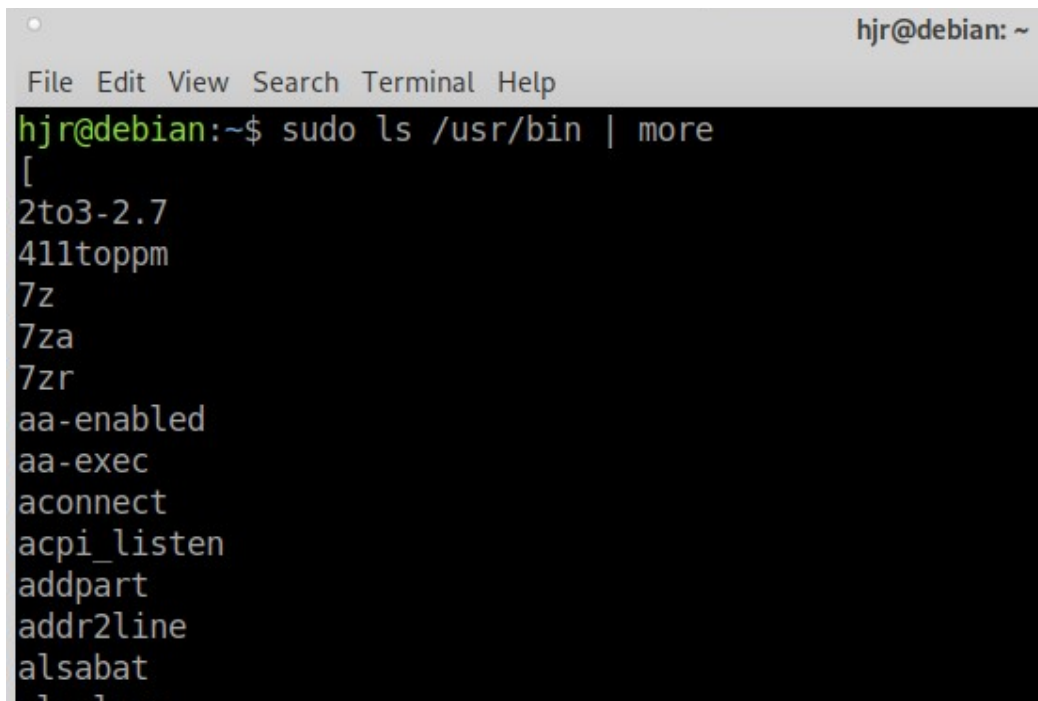
```
root    ALL=(ALL:ALL) ALL
```

... and type, **underneath it**, an equvalent line such as:

```
hjr     ALL=(ALL:ALL) ALL
```

The first part of that new line is your own username: in my case, it's 'hjr', but you replace that with whatever username you *actually* log onto your Linux PC with.

Then you save the modified file: in nano, you press Ctrl+X, then tap 'y' to confirm the save, and press Enter to save the modified file with the filename proposed. Job done: you can now stop being root (so, type **exit**). And now you can run any command with root privileges, just by prefixing the command with the word 'sudo':

This time, the 'ls /usr/bin' command produces no error at all.

As I say, there are only two occasions when Niente requires the use of sudo: installation and applying software updates, and both of those occasions are strictly 'one-off' or fairly rare affairs.

You definitely do NOT need sudo privileges to run Niente for the purposes of checking your own music collection for data integrity!

Acknowledgements