Music Organization

Take a music library and change its metatags and find duplicates.

- Music Duplicate Finder — Because Apparently You Collected the Same Song 47 Times
- dupfinder.py

Music Duplicate Finder — Because Apparently You Collected the Same Song 47 Times

So your music library is a disaster. You've got the same artist in two folders with slightly different names, three copies of the same track with different filenames, and somewhere in there a compilation from 2003 that contains literally every song you already own. Congratulations.

This guide walks through the full duplicate-finding and cleanup workflow in order. Each section is self-contained — if you only need one piece, jump to that section. If you're starting from scratch with a messy library, work through them in order.

Overview

Tool	What It Does
MusicBrainz Picard	Retags and reorganizes your library using accurate metadata
Beets	Finds exact duplicate tracks by artist and title
dupfinder.py	Finds near-duplicate tracks using partial title and artist name matching

---

Section 1 — MusicBrainz Picard

What It Does

Picard is a music tagger. It matches your files against the MusicBrainz database, corrects artist names, album names, track titles, genres, and track numbers, and can reorganize your folder structure to match. It does not remove duplicates on its own, but cleaning up your metadata first makes everything else in this guide work properly.

Installation

On Linux (recommended — gets you the latest version):

```
flatpak install flathub org.musicbrainz.Picard
```

On Linux via apt (older version but simpler):

```
sudo apt install musicbrainz-picard
```

On Windows/Mac: Download the installer from picard.musicbrainz.org

Flatpak Filesystem Access

If you installed via Flatpak and your music lives on a network share or non-standard path, you need to explicitly grant Picard access to that path or it will throw permission errors when saving:

```
flatpak override --user org.musicbrainz.Picard --filesystem=/path/to/your/music
```

Replace `/path/to/your/music` with the actual path where your music is stored. Restart Picard after running this.

Verify the override registered:

```
flatpak override --user --show org.musicbrainz.Picard
```

Configuration

Before loading any files, go to **Edit → Options** and configure the following:

Metadata tab

- Enable **Use genres from MusicBrainz**
- Enable **Fall back on album's artist genres if no genres are found**
- Set **Minimal genre usage** to 50–60% (90% is too strict and will tag very little)
- Set **Maximum number of genres** to 3–5

File Naming tab

- Enable **Rename files when saving**
- Enable **Move files when saving** and set the destination to your music folder
- Leave the default naming script as-is unless you have a preference — the default produces `Artist/Album/TrackNumber - Title`

Plugins tab

- Enable the **Genre** plugin if it is not already active

Running Picard

1. Open Picard and go to **File → Add Folder**, then select your music directory. All files will load into the left panel.
2. Click **Cluster** — groups files by folder into probable albums.
3. Click **Lookup** — matches clusters against MusicBrainz using existing tags.
4. Click **Scan** — runs a deeper pass on anything Lookup didn't confidently match.
5. Review the right panel. Color indicators mean:
   - **Green** — confident match, safe to save
   - **Yellow** — uncertain match, worth a quick look
   - **Red** — matched but with inconsistencies (missing tracks, wrong edition, etc.) — not necessarily wrong, just worth a glance
6. Anything still in the left panel after Scan couldn't be matched automatically. Right-click and choose **Search for similar releases** to find it manually, or leave it for later.
7. When satisfied, click **Save**. This is when files are actually renamed, moved, and retagged on disk.

Notes

- Picard does not delete duplicates. It may create them if the same song gets matched to two different releases. Deal with those in the next sections.
- If your music lives on a network share, saving will be slower than local disk. Let it finish — do not close Picard mid-save.
- Picard does not save session state. If you close it before saving, you start over from scratch. Yes, really.

Section 2 — Beets

What It Does

Beets finds exact duplicate tracks — same artist name, same track title — across your library. It works from its own internal database, so it needs to index your library before it can find anything.

Installation

```
sudo apt install beets
```

If the apt version is too old (check with `beet version` — anything below 1.6 is suspect):

```
sudo apt remove beets
pip install beets==1.6.0 --break-system-packages
```

Configuration

Create the config file if it does not exist:

```
nano ~/.config/beets/config.yaml
```

Add the following:

```
plugins: duplicates
directory: /path/to/your/music
library: ~/.config/beets/library.db
```

Replace `/path/to/your/music` with the actual path to your music folder.

Indexing Your Library

Beets needs to scan your library into its database before duplicate detection works. Run:

```
beet import -A /path/to/your/music
```

The `-A` flag imports everything without modifying tags or moving files — it only builds the index. This will take a while on a large library. Let it finish.

Verify it indexed correctly:

```
beet stats
```

That shows total track count, artists, albums, and library size. If the numbers look significantly off from what you expect, the import may not have finished.

Finding Duplicates

Find duplicates by artist and title:

```
beet duplicates -k title -k artist
```

This flags any tracks where both the artist name and title match exactly. Review the output before deleting anything.

When satisfied, delete the duplicates:

```
beet duplicates -k title -k artist -d
```

Limitations

Beets only catches **exact** matches. If one copy is tagged "The Spirit of Radio" and another is "The Spirit of Radio (live in Manchester)", Beets will not flag them as duplicates. For near-duplicate and fuzzy matching, use the script in Section 3.

Section 3 — dupfinder.py

What It Does

`dupfinder.py` is a Python script that catches duplicates Beets misses — specifically:

- Files in the same folder where only the track number prefix differs (`04 Title.mp3` vs `4-04 Title.mp3`)
- Files by the same artist where one title contains another (`The Spirit of Radio` vs `The Spirit of Radio (live in Manchester)`)
- Artist folders that are the same band under slightly different names (`Killers` vs `The Killers`)

It always runs in preview mode by default and never deletes anything without the `--delete` flag.

Requirements

- Python 3 (already installed on any modern Linux or Mac)
- The `dupfinder.py` script saved somewhere accessible — your home directory is fine

Running the Preview

Always start with a preview:

```
python3 /path/to/dupfinder.py /path/to/your/music
```

To save the output to a file so you can read it without it flying past your face:

```
python3 /path/to/dupfinder.py /path/to/your/music > ~/dupfinder_preview.txt
```

Open `dupfinder_preview.txt` in any text editor and review it.

Reading the Output

The output looks like this:

```
Artist: Rush
  KEEP:   The Spirit of Radio.mp3
          (/path/to/your/music/Rush/Permanent Waves/The Spirit of Radio.mp3)
  DELETE: The Spirit of Radio (live in Manchester).mp3
          (/path/to/your/music/Rush/Permanent Waves/The Spirit of Radio (live in
Manchester).mp3)
```

If artist name normalization merged two folders together, it will say so:

```
Artist (merged): Killers / The Killers
```

The script keeps the shorter, cleaner title and flags the longer variant for deletion. In most cases this is the right call — but read through the output anyway. You're the one who knows your library.

Deleting

When satisfied with the preview, run with `--delete`:

```
python3 /path/to/dupfinder.py /path/to/your/music --delete
```

After Deleting

Trigger a full rescan in your music server so it drops the deleted tracks from its database and stops pretending they still exist.

In Navidrome: **Settings → Scan Library → Full Rescan**

Notes

- Run these tools in order — Picard first, then Beets, then dupfinder.py. Clean metadata makes duplicate detection significantly more accurate.

- None of these tools touch your music server's database directly. They only operate on files on disk. Always rescan after making changes.
- If you want to check a new batch of incoming songs against your existing library without touching the library itself, see Scanning an Incoming Folder for Duplicates.

dupfinder.py

```
#!/usr/bin/env python3
"""
Music duplicate finder - combines three detection methods:

1. Track number prefix duplicates
   "4-04 The Spirit of Radio.mp3" vs "04 The Spirit of Radio.mp3"

2. Partial title matching within the same artist
   "The Spirit of Radio.mp3" vs "The Spirit of Radio (live in Manchester).mp3"

3. Artist name normalization
   "Killers" and "The Killers" treated as the same artist

Run without --delete to preview. Add --delete to remove duplicates.

Usage:
    python3 dupfinder.py /path/to/music
    python3 dupfinder.py /path/to/music --delete
"""

import os
import re
import argparse

AUDIO_EXTENSIONS = ('.mp3', '.flac', '.m4a', '.ogg', '.wav', '.aac')
TRACK_PREFIX = re.compile(r'^\d+[-\s]?\d*[-\s]?\s*')
ARTICLE_PREFIX = re.compile(r'^(the|a|an)\s+', re.IGNORECASE)



# ── Helpers ──────────────────────────────────────────────────────────

def strip_track_prefix(filename):
    """Remove leading track number from filename. Returns (bare_title, extension)."""
    name, ext = os.path.splitext(filename)
    stripped = TRACK_PREFIX.sub('', name).strip()
```

```
    return stripped, ext.lower()


def normalize_title(title):
    """Lowercase and strip punctuation for loose comparison."""
    return re.sub(r'[^\w\s]', '', title).lower().strip()


def normalize_artist(name):
    """Strip leading articles for artist folder comparison."""
    return ARTICLE_PREFIX.sub('', name).strip().lower()


def is_audio(filename):
    return filename.lower().endswith(AUDIO_EXTENSIONS)


# ── File collection ──────────────────────────────────────────────

def collect_files(folder):
    """Recursively collect all audio files under a folder."""
    files = []
    for root, dirs, filenames in os.walk(folder):
        for filename in filenames:
            if not is_audio(filename):
                continue
            bare, ext = strip_track_prefix(filename)
            files.append({
                'path': os.path.join(root, filename),
                'filename': filename,
                'bare': bare,
                'normalized': normalize_title(bare),
                'ext': ext,
            })
    return files


def group_artist_folders(music_dir):
    """
    Group artist-level folders by normalized name.
```

```
    e.g. 'Killers' and 'The Killers' end up in the same group.
    Returns a list of groups, each group being a list of folder paths.
    """
    folders = {}
    for entry in os.scandir(music_dir):
        if not entry.is_dir():
            continue
        key = normalize_artist(entry.name)
        folders.setdefault(key, []).append(entry.path)
    return list(folders.values())


# ─── Duplicate detection ────────────────────────────────────────────

def find_track_prefix_dupes(files):
    """
    Find files in the same folder where only the track number prefix differs.
    e.g. '04 Title.mp3' vs '4-04 Title.mp3'
    """
    by_folder = {}
    for f in files:
        folder = os.path.dirname(f['path'])
        by_folder.setdefault(folder, []).append(f)

    duplicates = []
    for folder, folder_files in by_folder.items():
        seen = {}
        for f in folder_files:
            key = (f['normalized'],)
            seen.setdefault(key, []).append(f)
        for key, group in seen.items():
            if len(group) > 1:
                duplicates.append(group)

    return duplicates


def find_partial_title_dupes(files):
    """
    Find files across the artist group where one title contains another.
```

```
    e.g. 'The Spirit of Radio' contained in 'The Spirit of Radio (live in Manchester)'
    Keeps the shorter/cleaner title.
    """
    duplicates = []
    used = set()

    sorted_files = sorted(files, key=lambda f: len(f['normalized']))

    for i, shorter in enumerate(sorted_files):
        if shorter['path'] in used:
            continue
        if not shorter['normalized']:
            continue

        group = [shorter]

        for longer in sorted_files[i + 1:]:
            if longer['path'] in used:
                continue
            if shorter['normalized'] in longer['normalized']:
                group.append(longer)
                used.add(longer['path'])

        if len(group) > 1:
            used.add(shorter['path'])
            duplicates.append(group)

    return duplicates


def pick_keeper_by_prefix(group):
    """For track prefix dupes, prefer simple '04 Title' over '4-04 Title'."""
    def score(f):
        if re.match(r'^\d+-\d+', f['filename']):
            return 1
        if re.match(r'^\d{2}\s', f['filename']):
            return 0
        return 2
    return sorted(group, key=score)
```

```
# ── Main ──────────────────────────────────────────────────────────────────

def main():
    parser = argparse.ArgumentParser(
        description='Find and remove duplicate music files.'
    )
    parser.add_argument('music_dir', help='Path to your music directory')
    parser.add_argument('--delete', action='store_true',
                        help='Actually delete duplicates (default is preview only)')
    args = parser.parse_args()

    music_dir = os.path.abspath(args.music_dir)
    print(f"Scanning: {music_dir}")
    print(f"Mode: {'DELETE' if args.delete else 'PREVIEW ONLY'}\n")

    artist_groups = group_artist_folders(music_dir)
    total_would_delete = 0
    total_deleted = 0

    for group_folders in sorted(artist_groups, key=lambda g: os.path.basename(g[0]).lower()):

        # Collect all files across all folders in this artist group
        all_files = []
        for folder in group_folders:
            all_files.extend(collect_files(folder))

        if not all_files:
            continue

        artist_label = ' / '.join(os.path.basename(f) for f in group_folders)

        # Run both detection passes
        prefix_dupes = find_track_prefix_dupes(all_files)
        partial_dupes = find_partial_title_dupes(all_files)

        all_dupes = prefix_dupes + partial_dupes

        if not all_dupes:
            continue
```

```
        # Flag if this group has multiple artist folders (name normalization hit)
        if len(group_folders) > 1:
            print(f"Artist (merged): {artist_label}")
        else:
            print(f"Artist: {artist_label}")

        for dup_group in all_dupes:
            sorted_group = pick_keeper_by_prefix(dup_group)
            keeper = sorted_group[0]
            to_delete = sorted_group[1:]

            print(f"  KEEP:   {keeper['bare']}{keeper['ext']}")
            print(f"          ({keeper['path']})")

            for dup in to_delete:
                print(f"  DELETE: {dup['bare']}{dup['ext']}")
                print(f"          ({dup['path']})")
                total_would_delete += 1

                if args.delete:
                    try:
                        os.remove(dup['path'])
                        total_deleted += 1
                    except Exception as e:
                        print(f"  ERROR: {e}")
        print()

    if args.delete:
        print(f"Done. Deleted {total_deleted} file(s).")
    else:
        print(f"Preview complete. {total_would_delete} file(s) would be deleted.")
        print("Run with --delete to actually remove them.")


if __name__ == '__main__':
    main()
```