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()
```

Revision #1
Created 2026-04-07 04:51:38 UTC by Danicus

Updated 2026-04-07 04:53:45 UTC by Danicus