normalize-transcript.py

```
#!/usr/bin/env python3
"""
normalize-transcript.py
Replaces alias variants in a whisper transcript with their canonical primary keys,
based on a TranscriptOMatic YAML meta file.

Usage:
    python3 normalize-transcript.py <transcript.txt> --game <slug> [--min-length N]

    Meta file is resolved relative to the transcript:
    <transcript-dir>/../../meta/<slug>.yaml
    Works on any machine regardless of the base directory name.

    If a matching .srt file exists alongside the .txt, it is normalized
    in sync. Borderline report is always generated from .txt only.

Output:
    <transcript_base>_normalized.txt     — cleaned transcript
    <transcript_base>_normalized.srt     — cleaned SRT (if .srt exists)
    <transcript_base>_borderline.txt     — borderline replacements for manual review

Options:
    --game SLUG      Game slug to resolve meta file (required)
    --min-length N   Minimum alias length to auto-replace (default: 5)
    --dry-run        Show what would be replaced without writing output
"""

import re
import sys
import yaml
import argparse
from pathlib import Path


MIN_LENGTH_DEFAULT = 5
```

```
def load_yaml(path):
    with open(path, encoding="utf-8") as f:
        return yaml.safe_load(f)


def extract_replacements(data, min_length=5):
    """
    Build two lists from the YAML:
    - replacements: [(alias, target), ...] for aliases >= min_length
    - borderline:   [(alias, target), ...] for aliases < min_length

    The replacement target is the entry's short: value if present,
    otherwise the primary key. This prevents partial-match duplication
    when primary keys contain substrings of each other.

    Covers: characters, groups, locations, terms, phrases, roles, players, gm
    """
    replacements = []
    borderline = []

    sections = [
        data.get("characters", {}),
        data.get("groups", {}),
        data.get("locations", {}),
        data.get("terms", {}),
        data.get("phrases", {}),
        data.get("players", {}),
        data.get("gm", {}),
        data.get("surnames", {}),
    ]

    for section in sections:
        if not isinstance(section, dict):
            continue
        for primary_key, entry in section.items():
            if not isinstance(entry, dict):
                continue
            aliases = entry.get("aliases", []) or []
```

```
            # Use short name as replacement target if available, else primary key.
            # This prevents partial-match duplication e.g. "Louis-Adrien de Bailly-Adrien de
Bailly".
            target = str(entry.get("short", primary_key) or primary_key)
            # safe: true   → force auto-replace, bypasses length check
            # safe: false  → borderline only, never auto-replace
            # safe: ignore → skip entirely, never replaced or reported
            # safe absent  → auto-replace if alias >= min_length, else borderline
            safe = entry.get("safe", None)
            for alias in aliases:
                if not alias or alias == target:
                    continue
                pair = (str(alias), target)
                if safe == "ignore":
                    continue
                elif safe is False:
                    borderline.append(pair)
                elif safe is True or len(str(alias)) >= min_length:
                    replacements.append(pair)
                else:
                    borderline.append(pair)

    # Roles section is a flat dict: role_name → character(s)
    # No aliases to replace here, skip.

    return replacements, borderline


def build_pattern(alias):
    """Word-boundary aware, case-insensitive regex for alias."""
    escaped = re.escape(alias)
    return re.compile(r'\b' + escaped + r'\b', re.IGNORECASE | re.UNICODE)


def normalize(text, replacements):
    """Apply all replacements to text.

    Longer aliases are processed first. Each match is immediately replaced
    with a unique placeholder so subsequent regexes cannot re-match already
    substituted text. Placeholders are resolved to their targets at the end.
```

```
    """
    sorted_replacements = sorted(replacements, key=lambda x: len(x[0]), reverse=True)
    # Use Private Use Area characters as placeholder delimiters —
    # vanishingly unlikely to appear in any real transcript.
    OPEN  = "\uE000"
    CLOSE = "\uE001"
    protected = []  # list of target strings, indexed by placeholder number

    for alias, target in sorted_replacements:
        pattern = build_pattern(alias)
        def replacer(m, t=target):
            idx = len(protected)
            protected.append(t)
            return f"{OPEN}{idx}{CLOSE}"
        text = pattern.sub(replacer, text)

    # Resolve placeholders in order
    for idx, target in enumerate(protected):
        text = text.replace(f"{OPEN}{idx}{CLOSE}", target)

    return text


def find_borderline_matches(lines, borderline):
    """Find lines containing borderline aliases and return report entries."""
    findings = []
    for lineno, line in enumerate(lines, 1):
        for alias, primary_key in borderline:
            pattern = build_pattern(alias)
            if pattern.search(line):
                findings.append((lineno, line.rstrip(), alias, primary_key))
    return findings


def main():
    parser = argparse.ArgumentParser(description="Normalize transcript using YAML meta file.")
    parser.add_argument("transcript", help="Path to transcript .txt file")
    parser.add_argument("--game", required=True, metavar="SLUG",
                        help="Game slug — resolves to META_DIR/<slug>.yaml")
    parser.add_argument("--min-length", type=int, default=MIN_LENGTH_DEFAULT,
```

```
                        help=f"Minimum alias length for auto-replacement (default:
{MIN_LENGTH_DEFAULT})")
    parser.add_argument("--dry-run", action="store_true",
                        help="Show replacements without writing output")
    args = parser.parse_args()

    transcript_path = Path(args.transcript)
    # Resolve meta dir relative to transcript: <session>/ → ../../meta/
    meta_path = (transcript_path.parent / ".." / ".." / "meta" /
f"{args.game}.yaml").resolve()

    if not transcript_path.exists():
        print(f"□ Transcript not found: {transcript_path}", file=sys.stderr)
        sys.exit(1)
    if not meta_path.exists():
        print(f"□ Meta file not found: {meta_path}", file=sys.stderr)
        print(f"   Expected: {meta_path}", file=sys.stderr)
        sys.exit(1)

    print(f"□□Transcript: {transcript_path}")
    print(f"□□Game:       {args.game}")
    print(f"□□Meta:       {meta_path}")
    print(f"□□Min alias length for auto-replace: {args.min_length}")
    print("----")

    data = load_yaml(str(meta_path))
    replacements, borderline = extract_replacements(data, args.min_length)

    print(f"□ {len(replacements)} aliases will be auto-replaced")
    print(f"⚠□  {len(borderline)} short/flagged aliases skipped (see report below)")
    print("----")

    # --- TXT ---
    text = transcript_path.read_text(encoding="utf-8")
    lines = text.splitlines()
    normalized_txt = normalize(text, replacements)

    # --- SRT (optional, normalized in sync with TXT) ---
    srt_path = transcript_path.with_suffix(".srt")
    srt_out_path = transcript_path.with_name(transcript_path.stem + "_normalized.srt")
```

```
has_srt = srt_path.exists()
if has_srt:
    normalized_srt = normalize(srt_path.read_text(encoding="utf-8"), replacements)

# --- Write output ---
if args.dry_run:
    print("□□Dry run — no files written.")
else:
    txt_out_path = transcript_path.with_name(transcript_path.stem + "_normalized.txt")
    txt_out_path.write_text(normalized_txt, encoding="utf-8")
    print(f"□ Written: {txt_out_path}")
    if has_srt:
        srt_out_path.write_text(normalized_srt, encoding="utf-8")
        print(f"□ Written: {srt_out_path}")
    else:
        print(f"□□  No matching .srt found alongside transcript — skipped.")

# --- Borderline report (from TXT only) ---
report_path = transcript_path.with_name(transcript_path.stem + "_borderline.txt")
if borderline:
    findings = find_borderline_matches(lines, borderline)
    if findings:
        header = (
            f"{'Line':<6} {'Alias':<20} {'Primary Key':<30} Context\n"
            f"{'----':<6} {'-----':<20} {'-----------':<30} -------\n"
        )
        rows = []
        for lineno, line, alias, primary_key in findings:
            context = line[:80] + ("…" if len(line) > 80 else "")
            rows.append(f"{lineno:<6} {alias:<20} {primary_key:<30} {context}")
        report_text = header + "\n".join(rows) + "\n"

        if not args.dry_run:
            report_path.write_text(report_text, encoding="utf-8")
            print(f"⚠□  Borderline report: {report_path} ({len(findings)} entries)")
        else:
            print("⚠□  Borderline replacements (dry run — not written):")
            print(header + "\n".join(rows))
    else:
        print("□ No borderline matches found in transcript.")
```

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

Created 2026-04-11 23:42:21 UTC by Mela
Updated 2026-04-12 11:02:48 UTC by Mela