The Scripts

- build-whisper-prompt.py
- transcribe-audio.sh - mit Prompt
- normalize-transcript.py
- Attic
  - transcribe_audio.sh — Pre-Context
  - transcribe-audio.sh - With Context
- summarize-meeting.sh
- build-summary-context.py

build-whisper-prompt.py

```
#!/usr/bin/env python3
"""
build-whisper-prompt.py
Extracts primary keys from a TranscriptOMatic YAML meta file and writes
them one per line to a .prompt file for use as a whisper vocabulary hint.

The output is a starting point — edit it manually to remove common words
that don't benefit from hinting and to stay within Whisper's ~224 token limit.

Usage:
    python3 build-whisper-prompt.py --game <slug>

    YAML:   resolved from <script-dir>/../meta/<slug>.yaml
    Output: <script-dir>/../meta/<slug>.prompt

    Run from anywhere; paths are relative to the script location.

Options:
    --game SLUG     Game slug (required)
    --force         Overwrite existing .prompt file (default: abort if exists)
"""

import sys
import yaml
import argparse
from pathlib import Path



# Sections to extract primary keys from, in priority order.
# Terms and locations first — most phonetically unusual for Whisper.
SECTIONS = ["terms", "surnames", "locations", "characters", "groups", "phrases", "players",
"gm"]



def load_yaml(path):
```

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


def extract_keys(data):
    """Extract primary keys and titles from all relevant sections."""
    keys = []
    seen = set()

    def add(term):
        clean = str(term).split("(")[0].strip()
        if clean and clean not in seen:
            keys.append(clean)
            seen.add(clean)

    for section in SECTIONS:
        block = data.get(section, {}) or {}
        if not isinstance(block, dict):
            continue
        for key, entry in block.items():
            add(key)
            if isinstance(entry, dict):
                # Titles: phonetically unusual, benefit from hinting
                for title in (entry.get("titles") or []):
                    add(title)
                # English name: intentional alternate identity, include for Whisper awareness
                name_en = entry.get("name_en")
                if name_en:
                    add(name_en)
    return keys


def main():
    script_dir = Path(__file__).resolve().parent
    meta_dir = (script_dir / ".." / "meta").resolve()

    parser = argparse.ArgumentParser(
        description="Generate a whisper vocabulary prompt file from a YAML meta file."
    )
    parser.add_argument("--game", required=True, metavar="SLUG",
```

```
                        help="Game slug — resolves to meta/<slug>.yaml")
    parser.add_argument("--force", action="store_true",
                        help="Overwrite existing .prompt file")
    args = parser.parse_args()

    yaml_path = meta_dir / f"{args.game}.yaml"
    prompt_path = meta_dir / f"{args.game}.prompt"

    if not yaml_path.exists():
        print(f"□ YAML not found: {yaml_path}", file=sys.stderr)
        sys.exit(1)

    if prompt_path.exists() and not args.force:
        print(f"□ Prompt file already exists: {prompt_path}", file=sys.stderr)
        print( "   Use --force to overwrite.", file=sys.stderr)
        sys.exit(1)

    data = load_yaml(yaml_path)
    keys = extract_keys(data)

    prompt_path.write_text("\n".join(keys) + "\n", encoding="utf-8")

    print(f"□□YAML:   {yaml_path}")
    print(f"□ Written: {prompt_path}")
    print(f"   {len(keys)} terms — edit to remove unproblematic entries")
    print(f"   then check token count (target: <224 tokens)")


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

transcribe-audio.sh - mit Prompt

```
#!/usr/bin/env bash
set -euo pipefail

# ------------------------------------------------------------
# transcribe-audio.sh
# Post-session transcription on Mac using whisper-cli
# Usage: transcribe-audio.sh [--de|--en|--auto] [--game <slug>]
#                            [--vad-preset tight|default|loose]
#                            [--vt F] [--vspd N] [--vp N] [--et F] [--nth F]
#                            <audio.wav>
# ------------------------------------------------------------

WHISPER="$HOME/Transkriptionen/whisper.cpp/build/bin/whisper-cli"
MODEL_EN="$HOME/Transkriptionen/whisper.cpp/models/ggml-large-v3-turbo.bin"
MODEL_DE="$HOME/Transkriptionen/whisper.cpp/models/ggml-large-v3-turbo-german.bin"
VAD_MODEL="$HOME/Transkriptionen/whisper.cpp/models/ggml-silero-v6.2.0.bin"

# ------------------------------------------------------------
# Argument parsing
# ------------------------------------------------------------

LANG_MODE="en"
AUDIO=""
GAME_SLUG=""
VAD_PRESET=""   # empty = use language default
VAD_VT=""       # override: voice threshold
VAD_VSPD=""     # override: min speech duration (ms)
VAD_VP=""       # override: padding (ms)
VAD_ET=""       # override: end-of-speech timeout
VAD_NTH=""      # override: noise threshold

while [[ $# -gt 0 ]]; do
```

```
  case "$1" in
    --de)          LANG_MODE="de";    shift ;;
    --en)          LANG_MODE="en";    shift ;;
    --auto)        LANG_MODE="auto";  shift ;;
    --game)        GAME_SLUG="$2";    shift 2 ;;
    --vad-preset)  VAD_PRESET="$2";   shift 2 ;;
    --vt)          VAD_VT="$2";       shift 2 ;;
    --vspd)        VAD_VSPD="$2";     shift 2 ;;
    --vp)          VAD_VP="$2";       shift 2 ;;
    --et)          VAD_ET="$2";       shift 2 ;;
    --nth)         VAD_NTH="$2";      shift 2 ;;
    -*)
      echo "Usage: transcribe-audio [--de|--en|--auto] [--game <slug>]" >&2
      echo "       [--vad-preset tight|default|loose]" >&2
      echo "       [--vt F] [--vspd N] [--vp N] [--et F] [--nth F]" >&2
      echo "" >&2
      echo "VAD presets:" >&2
      echo "  tight    vt=0.30 vspd=250 vp=400 et=2.3 nth=0.40  fast speakers, good mic" >&2
      echo "  default  vt=0.25 vspd=150 vp=200 et=2.8 nth=0.30  mixed tempo, moderate pauses"
>&2
      echo "  loose    vt=0.20 vspd=100 vp=600 et=3.5 nth=0.20  slow speakers, noisy room" >&2
      exit 2
      ;;
    *)
      AUDIO="$1"; shift ;;
  esac
done

if [[ -z "$AUDIO" ]]; then
  echo "□ No audio file specified." >&2
  echo "   Usage: transcribe-audio [--de|--en|--auto] [--game <slug>] <audio.wav>" >&2
  exit 1
fi

if [[ ! -f "$AUDIO" ]]; then
  echo "□ File not found: $AUDIO" >&2
  exit 1
fi

# ------------------------------------------------------------
```

```
# Language options, model selection, and VAD parameters
# ------------------------------------------------------------

# Language → model + default preset
case "$LANG_MODE" in
  en)
    LANG_OPTS=(-l en)
    MODEL="$MODEL_EN"
    [[ -z "$VAD_PRESET" ]] && VAD_PRESET="tight"
    ;;
  de)
    LANG_OPTS=(-l de)
    MODEL="$MODEL_DE"
    [[ -z "$VAD_PRESET" ]] && VAD_PRESET="default"
    ;;
  auto)
    LANG_OPTS=()
    MODEL="$MODEL_EN"
    [[ -z "$VAD_PRESET" ]] && VAD_PRESET="default"
    ;;
esac

# Preset base values
#   tight    fast speakers, few pauses, good mic        (EN default)
#   default  mixed tempo, moderate pauses               (DE / auto default)
#   loose    slow/deliberate speakers, noisier room
case "$VAD_PRESET" in
  tight)   _VT=0.30; _VSPD=250; _VP=400; _ET=2.3; _NTH=0.40 ;;
  default) _VT=0.25; _VSPD=150; _VP=200; _ET=2.8; _NTH=0.30 ;;
  loose)   _VT=0.20; _VSPD=100; _VP=600; _ET=3.5; _NTH=0.20 ;;
  *)
    echo "□ Unknown VAD preset: '$VAD_PRESET'. Use tight, default, or loose." >&2
    exit 2
    ;;
esac

# Apply per-parameter overrides
VT="${VAD_VT:-$_VT}"
VSPD="${VAD_VSPD:-$_VSPD}"
VP="${VAD_VP:-$_VP}"
```

```
ET="${VAD_ET:-$_ET}"
NTH="${VAD_NTH:-$_NTH}"

VAD_OPTS=(-vt "$VT" -vspd "$VSPD" -vp "$VP" -et "$ET" -nth "$NTH")

# ------------------------------------------------------------
# Build prompt from .prompt file (if --game given)
# ------------------------------------------------------------

PROMPT_OPTS=()

if [[ -z "$GAME_SLUG" ]]; then
  echo "⚠  No --game specified. Running without vocabulary prompt."
  echo "   Tip: use --game <slug> for better transcription of proper nouns."
fi

if [[ -n "$GAME_SLUG" ]]; then
  PROMPT_FILE="$(cd "$(dirname "$AUDIO")" && cd ../../meta && pwd)/${GAME_SLUG}.prompt"
  if [[ -f "$PROMPT_FILE" ]]; then
    # Join lines into comma-separated list for whisper --prompt
    PROMPT_TEXT="$(grep -v '^\s*$' "$PROMPT_FILE" | paste -sd ',' - | sed 's/,/, /g')"
    if [[ -n "$PROMPT_TEXT" ]]; then
      PROMPT_OPTS=(--prompt "$PROMPT_TEXT" --carry-initial-prompt)
      echo "□□Game:       $GAME_SLUG"
      echo "□□Prompt:     $PROMPT_TEXT"
    fi
  else
    echo "⚠  No prompt file found: $PROMPT_FILE" >&2
    echo "   Run: build-whisper-prompt.py --game $GAME_SLUG" >&2
  fi
fi

# ------------------------------------------------------------
# Output path: same dir as audio, transcript suffix
# ------------------------------------------------------------

AUDIO_DIR="$(dirname "$AUDIO")"
AUDIO_BASE="$(basename "$AUDIO" .wav)"
TRANSCRIPT_BASE="$AUDIO_DIR/${AUDIO_BASE}_transcript"
```

```
echo "□□Audio:      $AUDIO"
echo "□□□ Language:   $LANG_MODE | Model: $(basename "$MODEL")"
echo "   VAD preset: $VAD_PRESET | vt=$VT  vspd=$VSPD  vp=$VP  et=$ET  nth=$NTH"
echo "□□Transcript: ${TRANSCRIPT_BASE}.txt"
echo "----"

# ------------------------------------------------------------
# Run whisper-cli
# ------------------------------------------------------------

"$WHISPER" \
  -m "$MODEL" \
  "${LANG_OPTS[@]}" \
  --vad \
  -vm "$VAD_MODEL" \
  --output-txt \
  --output-srt \
  -of "$TRANSCRIPT_BASE" \
  "${VAD_OPTS[@]}" \
  "${PROMPT_OPTS[@]}" \
  -f "$AUDIO"

# Strip ANSI escape sequences and carriage returns from txt output
sed -i '' 's/\x1b\[[0-9;]*[mGKH]//g; s/\r//g' "${TRANSCRIPT_BASE}.txt"

echo "□ Done: ${TRANSCRIPT_BASE}.txt"
```

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()
```

Attic

transcribe_audio.sh — Pre-Context

```
#!/usr/bin/env bash
set -euo pipefail

# ------------------------------------------------------------
# transcribe-audio.sh
# Post-session transcription on Mac using whisper-cli
# Usage: transcribe-audio.sh [--de|--en|--auto] <audio.wav>
# ------------------------------------------------------------

WHISPER="$HOME/Transkriptionen/whisper.cpp/build/bin/whisper-cli"
MODEL="$HOME/Transkriptionen/whisper.cpp/models/ggml-large-v3-turbo.bin"
VAD_MODEL="$HOME/Transkriptionen/whisper.cpp/models/ggml-silero-v6.2.0.bin"

# ------------------------------------------------------------
# Argument parsing
# ------------------------------------------------------------

LANG_MODE="en"
AUDIO=""

while [[ $# -gt 0 ]]; do
  case "$1" in
    --de)   LANG_MODE="de"; shift ;;
    --en)   LANG_MODE="en"; shift ;;
    --auto) LANG_MODE="auto"; shift ;;
    -*)
      echo "Usage: transcribe-audio [--de|--en|--auto] <audio.wav>" >&2
      exit 2
      ;;
    *)
```

```
      AUDIO="$1"; shift ;;
  esac
done

if [[ -z "$AUDIO" ]]; then
  echo "□ No audio file specified." >&2
  echo "   Usage: transcribe-audio [--de|--en|--auto] <audio.wav>" >&2
  exit 1
fi

if [[ ! -f "$AUDIO" ]]; then
  echo "□ File not found: $AUDIO" >&2
  exit 1
fi

# ------------------------------------------------------------
# Language options
# ------------------------------------------------------------

case "$LANG_MODE" in
  en|de) LANG_OPTS=(-l "$LANG_MODE") ;;
  auto)  LANG_OPTS=() ;;
esac

# ------------------------------------------------------------
# Output path: same dir as audio, .txt extension
# ------------------------------------------------------------

AUDIO_DIR="$(dirname "$AUDIO")"
AUDIO_BASE="$(basename "$AUDIO" .wav)"
TRANSCRIPT_BASE="$AUDIO_DIR/${AUDIO_BASE}_transcript"

echo "□□Audio:      $AUDIO"
echo "□□□ Language:   $LANG_MODE"
echo "□□Transcript: ${TRANSCRIPT_BASE}.txt"
echo "----"

# ------------------------------------------------------------
# Run whisper-cli
# ------------------------------------------------------------
```

```
  auto)  LANG_OPTS=() ;;
esac

# ------------------------------------------------------------
# Output path: same dir as audio, .txt extension
# ------------------------------------------------------------

AUDIO_DIR="$(dirname "$AUDIO")"
AUDIO_BASE="$(basename "$AUDIO" .wav)"
TRANSCRIPT_BASE="$AUDIO_DIR/${AUDIO_BASE}_transcript"

echo "□□Audio:      $AUDIO"
echo "□□□ Language:   $LANG_MODE"
echo "□□Transcript: ${TRANSCRIPT_BASE}.txt"
echo "----"

# ------------------------------------------------------------
# Run whisper-cli
# ------------------------------------------------------------

"$WHISPER" \
  -m "$MODEL" \
  "${LANG_OPTS[@]}" \
  --vad \
  -vm "$VAD_MODEL" \
  --output-txt \
  --output-srt \
  -of "$TRANSCRIPT_BASE" \
  -f "$AUDIO"


# Strip ANSI escape sequences and carriage returns from txt output
sed -i '' 's/\x1b\[[0-9;]*[mGKH]//g; s/\r//g' "${TRANSCRIPT_BASE}.txt"

echo "□ Done: ${TRANSCRIPT_BASE}.txt"
```

transcribe-audio.sh - With Context

```
#!/usr/bin/env bash
set -euo pipefail

# ------------------------------------------------------------
# transcribe-audio.sh
# Post-session transcription on Mac using whisper-cli
# Usage: transcribe-audio.sh [--de|--en|--auto] [--game <slug>]
#                            [--vad-preset tight|default|loose]
#                            [--vt F] [--vspd N] [--vp N] [--et F] [--nth F]
#                            <audio.wav>
# ------------------------------------------------------------

WHISPER="$HOME/Transkriptionen/whisper.cpp/build/bin/whisper-cli"
MODEL="$HOME/Transkriptionen/whisper.cpp/models/ggml-large-v3-turbo.bin"
VAD_MODEL="$HOME/Transkriptionen/whisper.cpp/models/ggml-silero-v6.2.0.bin"
META_DIR="$HOME/Syncthing/TranscriptOMatic/meta"

# ------------------------------------------------------------
# Argument parsing
# ------------------------------------------------------------

LANG_MODE="en"
AUDIO=""
GAME_SLUG=""
VAD_PRESET=""   # empty = use language default
VAD_VT=""       # override: voice threshold
VAD_VSPD=""     # override: min speech duration (ms)
VAD_VP=""       # override: padding (ms)
VAD_ET=""       # override: end-of-speech timeout
VAD_NTH=""      # override: noise threshold
```

```
while [[ $# -gt 0 ]]; do
  case "$1" in
    --de)          LANG_MODE="de";    shift ;;
    --en)          LANG_MODE="en";    shift ;;
    --auto)        LANG_MODE="auto";  shift ;;
    --game)        GAME_SLUG="$2";    shift 2 ;;
    --vad-preset)  VAD_PRESET="$2";   shift 2 ;;
    --vt)          VAD_VT="$2";       shift 2 ;;
    --vspd)        VAD_VSPD="$2";     shift 2 ;;
    --vp)          VAD_VP="$2";       shift 2 ;;
    --et)          VAD_ET="$2";       shift 2 ;;
    --nth)         VAD_NTH="$2";      shift 2 ;;
    -*)
      echo "Usage: transcribe-audio [--de|--en|--auto] [--game <slug>]" >&2
      echo "       [--vad-preset tight|default|loose]" >&2
      echo "       [--vt F] [--vspd N] [--vp N] [--et F] [--nth F]" >&2
      echo "" >&2
      echo "VAD presets:" >&2
      echo "  tight    vt=0.30 vspd=250 vp=400 et=2.3 nth=0.40  fast speakers, good mic" >&2
      echo "  default  vt=0.25 vspd=150 vp=200 et=2.8 nth=0.30  mixed tempo, moderate pauses"
>&2
      echo "  loose    vt=0.20 vspd=100 vp=600 et=3.5 nth=0.20  slow speakers, noisy room" >&2
      exit 2
      ;;
    *)
      AUDIO="$1"; shift ;;
  esac
done

if [[ -z "$AUDIO" ]]; then
  echo "□ No audio file specified." >&2
  echo "   Usage: transcribe-audio [--de|--en|--auto] [--game <slug>] <audio.wav>" >&2
  exit 1
fi

if [[ ! -f "$AUDIO" ]]; then
  echo "□ File not found: $AUDIO" >&2
  exit 1
fi
```

```
# ------------------------------------------------------------
# Language options and VAD parameters
# ------------------------------------------------------------

# Language → default preset
case "$LANG_MODE" in
  en)
    LANG_OPTS=(-l en)
    [[ -z "$VAD_PRESET" ]] && VAD_PRESET="tight"
    ;;
  de)
    LANG_OPTS=(-l de)
    [[ -z "$VAD_PRESET" ]] && VAD_PRESET="default"
    ;;
  auto)
    LANG_OPTS=()
    [[ -z "$VAD_PRESET" ]] && VAD_PRESET="default"
    ;;
esac

# Preset base values
#   tight    fast speakers, few pauses, good mic        (EN default)
#   default  mixed tempo, moderate pauses               (DE / auto default)
#   loose    slow/deliberate speakers, noisier room
case "$VAD_PRESET" in
  tight)   _VT=0.30; _VSPD=250; _VP=400; _ET=2.3; _NTH=0.40 ;;
  default) _VT=0.25; _VSPD=150; _VP=200; _ET=2.8; _NTH=0.30 ;;
  loose)   _VT=0.20; _VSPD=100; _VP=600; _ET=3.5; _NTH=0.20 ;;
  *)
    echo "□ Unknown VAD preset: '$VAD_PRESET'. Use tight, default, or loose." >&2
    exit 2
    ;;
esac

# Apply per-parameter overrides
VT="${VAD_VT:-$_VT}"
VSPD="${VAD_VSPD:-$_VSPD}"
VP="${VAD_VP:-$_VP}"
ET="${VAD_ET:-$_ET}"
```

```
NTH="${VAD_NTH:-$_NTH}"

VAD_OPTS=(-vt "$VT" -vspd "$VSPD" -vp "$VP" -et "$ET" -nth "$NTH")

# ------------------------------------------------------------
# Build prompt from YAML meta file (if --game given)
# ------------------------------------------------------------

PROMPT_OPTS=()

if [[ -z "$GAME_SLUG" ]]; then
  echo "⚠  No --game specified. Running without vocabulary prompt."
  echo "   Tip: use --game <slug> for better transcription of proper nouns."
fi

if [[ -n "$GAME_SLUG" ]]; then
  META_FILE="$META_DIR/${GAME_SLUG}.yaml"
  if [[ -f "$META_FILE" ]]; then
    # Extract all primary keys from characters, locations, terms, phrases, groups
    # using Python — handles Unicode correctly
    PROMPT_TEXT="$(uv run --with pyyaml python3 - "$META_FILE" <<'PYEOF'
import sys, yaml

with open(sys.argv[1], encoding="utf-8") as f:
    data = yaml.safe_load(f)

keys = []
for section in ["characters", "locations", "terms", "phrases", "groups"]:
    block = data.get(section, {}) or {}
    if isinstance(block, dict):
        for key in block.keys():
            # Strip parenthetical suffixes e.g. "Muiris Doyle (Ó Dubhghaill)"
            clean = key.split("(")[0].strip()
            if clean:
                keys.append(clean)

print(", ".join(keys))
PYEOF
)"
    if [[ -n "$PROMPT_TEXT" ]]; then
```

```
      PROMPT_OPTS=(--prompt "$PROMPT_TEXT" --carry-initial-prompt)
      echo "□□Game:       $GAME_SLUG"
      echo "□□Prompt:     $PROMPT_TEXT"
    fi
  else
    echo "⚠□  No meta file found for slug '$GAME_SLUG' in $META_DIR" >&2
  fi
fi

# ------------------------------------------------------------
# Output path: same dir as audio, transcript suffix
# ------------------------------------------------------------

AUDIO_DIR="$(dirname "$AUDIO")"
AUDIO_BASE="$(basename "$AUDIO" .wav)"
TRANSCRIPT_BASE="$AUDIO_DIR/${AUDIO_BASE}_transcript"

echo "□□Audio:      $AUDIO"
echo "□□□ Language:   $LANG_MODE | VAD preset: $VAD_PRESET"
echo "   vt=$VT  vspd=$VSPD  vp=$VP  et=$ET  nth=$NTH"
echo "□□Transcript: ${TRANSCRIPT_BASE}.txt"
echo "----"

# ------------------------------------------------------------
# Run whisper-cli
# ------------------------------------------------------------

"$WHISPER" \
  -m "$MODEL" \
  "${LANG_OPTS[@]}" \
  --vad \
  -vm "$VAD_MODEL" \
  --output-txt \
  --output-srt \
  -of "$TRANSCRIPT_BASE" \
  "${VAD_OPTS[@]}" \
  "${PROMPT_OPTS[@]}" \
  -f "$AUDIO"

# Strip ANSI escape sequences and carriage returns from txt output
```

```
sed -i '' 's/\x1b\[[0-9;]*[mGKH]//g; s/\r//g' "${TRANSCRIPT_BASE}.txt"

echo "□ Done: ${TRANSCRIPT_BASE}.txt"
```

summarize-meeting.sh

```
#!/usr/bin/env bash
set -euo pipefail

# ------------------------------------------------------------
# summarize-meeting.sh
# Post-session summarization on Mac using ollama
# Usage: summarize-meeting.sh [--de|--en] [--game <slug>] [<transcript.txt>]
#        Defaults to English, most recent transcript if none given
# ------------------------------------------------------------

OLLAMA_MODEL="${OLLAMA_MODEL:-gemma3:27b}"
BASE="$HOME/Syncthing/TranscriptOMatic/recordings"
META_DIR="$(cd "$(dirname "$0")/../meta" 2>/dev/null && pwd || echo
"$HOME/Syncthing/TranscriptOMatic/meta")"
LANG_MODE="en"
TRANSCRIPT=""
GAME_SLUG=""
USE_CONTEXT=0

# ------------------------------------------------------------
# Argument parsing
# ------------------------------------------------------------

while [[ $# -gt 0 ]]; do
  case "$1" in
    --de)     LANG_MODE="de"; shift ;;
    --en)     LANG_MODE="en"; shift ;;
    --game)       GAME_SLUG="$2"; shift 2 ;;
    --context)    USE_CONTEXT=1; shift ;;
    -*)
      echo "Usage: summarize-meeting [--de|--en] [--game <slug>] [--context]
[<transcript.txt>]" >&2
      exit 2
      ;;
    *)
```

```
      TRANSCRIPT="$1"; shift ;;
  esac
done

# ------------------------------------------------------------
# Find transcript
# ------------------------------------------------------------

if [[ -z "$TRANSCRIPT" ]]; then
  # Prefer normalized transcript; fall back to plain transcript
  TRANSCRIPT="$(ls -t "$BASE"/**/*_normalized.txt 2>/dev/null | head -n1 || true)"
  if [[ -z "$TRANSCRIPT" ]]; then
    TRANSCRIPT="$(ls -t "$BASE"/**/*_transcript.txt 2>/dev/null | head -n1 || true)"
  fi
  if [[ -z "$TRANSCRIPT" ]]; then
    echo "□ No transcript found in $BASE" >&2
    echo "   Usage: summarize-meeting [--de|--en] [--game <slug>] [<transcript.txt>]" >&2
    exit 1
  fi
fi

if [[ ! -f "$TRANSCRIPT" ]]; then
  echo "□ File not found: $TRANSCRIPT" >&2
  exit 1
fi

# If a plain transcript was given explicitly, check whether a normalized version exists
if [[ "$TRANSCRIPT" == *_transcript.txt ]]; then
  NORMALIZED="${TRANSCRIPT/_transcript.txt/_transcript_normalized.txt}"
  if [[ -f "$NORMALIZED" ]]; then
    echo "□□  Using normalized transcript: $NORMALIZED"
    TRANSCRIPT="$NORMALIZED"
  fi
fi

# ------------------------------------------------------------
# Auto-detect game slug from transcript filename if not given
# ------------------------------------------------------------

if [[ -z "$GAME_SLUG" ]]; then
```

```
  TRANSCRIPT_BASENAME="$(basename "$TRANSCRIPT")"
  # Try to find a matching meta file by checking if any slug appears in the filename
  for META_FILE in "$META_DIR"/*.yaml; do
    [[ -f "$META_FILE" ]] || continue
    CANDIDATE_SLUG="$(basename "$META_FILE" .yaml)"
    if [[ "$TRANSCRIPT_BASENAME" == *"$CANDIDATE_SLUG"* ]]; then
      GAME_SLUG="$CANDIDATE_SLUG"
      break
    fi
  done
fi

# ------------------------------------------------------------
# Load meta file if available
# ------------------------------------------------------------

META_CONTEXT=""
META_FILE=""

if [[ -n "$GAME_SLUG" && "$USE_CONTEXT" == "1" ]]; then
  META_FILE="$META_DIR/${GAME_SLUG}.yaml"
  if [[ -f "$META_FILE" ]]; then
    META_CONTEXT="$(uv run --with pyyaml python3 "$(dirname "$0")/build-summary-context.py" --
game "$GAME_SLUG")"
    echo "□□Meta:       $META_FILE (filtered)"
  else
    echo "⚠□  No meta file found for slug '$GAME_SLUG' in $META_DIR" >&2
  fi
else
  echo "□□  Running without context document (use --context to include)"
fi

# ------------------------------------------------------------
# Output paths
# ------------------------------------------------------------

SESSION="$(dirname "$TRANSCRIPT")"
TRANSCRIPT_BASE="$(basename "$TRANSCRIPT" .txt)"
SUMMARY="$SESSION/${TRANSCRIPT_BASE}_summary.md"
```

```
# ------------------------------------------------------------
# Skip if summary already exists (use FORCE=1 to override)
# ------------------------------------------------------------

if [[ -f "$SUMMARY" && "${FORCE:-}" != "1" ]]; then
  echo "□□  Summary already exists, skipping: $SUMMARY"
  echo "   Use FORCE=1 summarize-meeting to overwrite."
  exit 0
fi

# ------------------------------------------------------------
# Language-specific prompt
# ------------------------------------------------------------

case "$LANG_MODE" in
  en) PROMPT_LANG="Write the summary in English." ;;
  de) PROMPT_LANG="Schreibe die Zusammenfassung auf Deutsch." ;;
esac

echo "□□Transcript: $TRANSCRIPT"
echo "□□Model:      $OLLAMA_MODEL"
echo "□□□ Language:   $LANG_MODE"
echo "□□Summary:    $SUMMARY"
echo "----"

# ------------------------------------------------------------
# Build prompt
# ------------------------------------------------------------

if [[ -n "$META_CONTEXT" ]]; then
  CONTEXT_BLOCK="REFERENCE DOCUMENT (metadata only — not part of the session transcript):
This YAML document provides background context for correctly interpreting the transcript
below.
It is NOT a session log or discussion. Do NOT summarize or reference its contents as in-game
or out-of-game events.
Use it exclusively to:
- Correctly spell and identify character names, player names, locations and in-game terms
- Understand roles, group memberships and character relationships
- The 'lang' field (e.g. ga, fr) indicates language origin of a name — it is metadata, not a
topic of discussion
```

```
- Use the 'short' name for characters in continuous prose; use full names only when
introducing a character
- The 'aliases' field lists transcription variants of a name — use only the primary key or
short name in the summary
- If a character appears in the transcript under an alias, identify them by their short name

Reference document:
---
${META_CONTEXT}
---
END OF REFERENCE DOCUMENT

The transcript follows below. Summarize only what is in the transcript.
"
else
  CONTEXT_BLOCK=""
fi

# ------------------------------------------------------------
# Run summarization
# ------------------------------------------------------------

ollama --nowordwrap run "$OLLAMA_MODEL" <<EOF > "$SUMMARY"
You are an expert note-taker for tabletop roleplaying game sessions.
The transcript is a recording of a TTRPG session and contains both in-character roleplay and
out-of-character table talk.

${CONTEXT_BLOCK}
Rules:
- Clearly distinguish between in-character events and out-of-character discussion
- Quote spoken statements in their original language
- ${PROMPT_LANG}
- Use character short names (as provided in context) rather than full names where natural
- ONLY summarize events, decisions and facts that are explicitly stated in the transcript
- If you are not certain something happened, omit it entirely — do NOT infer, imply or
extrapolate
- Pay close attention to who does what: do not invert agency (e.g. who challenges whom, who
saves whom)
- Do not conflate events from different sessions; only summarize what happens in this
transcript
```

```
- Subtext and player motivation matter: note what characters say and do, not what the model thinks they mean

Deliver:
1) Session overview (3-5 sentences summarising the main in-game events)
2) Key in-game decisions and developments
3) Important character moments (emotional beats, revelations, relationship shifts)
4) Highlights (memorable quotes, unexpected twists, standout scenes)
5) Notable out-of-character moments (rules discussions, retcons, player notes)
6) Cliffhangers and open threads going into the next session
7) Characters introduced or significantly developed this session

Transcript:
$(cat "$TRANSCRIPT")
EOF

echo "□ Summary written to $SUMMARY"
```

build-summary-context.py

```
#!/usr/bin/env python3
"""
build-summary-context.py
Extracts summarizer-relevant fields from a TranscriptOMatic YAML meta file,
stripping transcription-specific fields (aliases, safe, lang, deedname, etc.)
that confuse LLMs when used as summary context.

Output is plain text, structured for readability by an LLM.

Usage:
    python3 build-summary-context.py --game <slug>

    YAML resolved from <script-dir>/../meta/<slug>.yaml
    Output printed to stdout (pipe into summarize-meeting.sh)

Options:
    --game SLUG     Game slug (required)
"""

import sys
import yaml
import argparse
from pathlib import Path


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


def format_list(value):
    if not value:
        return None
    if isinstance(value, str):
        return value.strip() or None
```

```
    if isinstance(value, list):
        items = [str(v).strip() for v in value if v]
        return ", ".join(items) if items else None
    return str(value).strip() or None


def build_context(data):
    lines = []

    # Game info
    game = data.get("game", "")
    system = data.get("system", "")
    setting = (data.get("setting") or "").strip()
    if game:
        lines.append(f"Game: {game}")
    if system:
        lines.append(f"System: {system}")
    if setting:
        lines.append(f"Setting: {setting}")
    if data.get("notes"):
        lines.append(f"Notes: {data['notes'].strip()}")
    lines.append("")

    # Characters
    characters = data.get("characters", {}) or {}
    if characters:
        lines.append("== CHARACTERS ==")
        for primary_key, entry in characters.items():
            if not isinstance(entry, dict):
                continue
            short = entry.get("short") or ""
            name_en = format_list(entry.get("name_en"))
            titles = format_list(entry.get("titles"))
            roles = format_list(entry.get("roles"))
            groups = format_list(entry.get("groups"))

            # Build display name
            display = primary_key
            if short and short != primary_key:
                display = f"{primary_key} (called: {short})"
```

```
        if name_en:
            display += f" / also known as: {name_en}"

        parts = [f"- {display}"]
        if titles:
            parts.append(f"  Titles: {titles}")
        if roles:
            parts.append(f"  Role: {roles}")
        if groups:
            parts.append(f"  Groups: {groups}")
        lines.extend(parts)
    lines.append("")

# Players and GM (out-of-character)
gm = data.get("gm", {}) or {}
players = data.get("players", {}) or {}
ooc = {}
if isinstance(gm, dict):
    ooc.update(gm)
if isinstance(players, dict):
    ooc.update(players)
if ooc:
    lines.append("== PLAYERS (out-of-character) ==")
    for name in ooc.keys():
        lines.append(f"- {name}")
    lines.append("")

# Groups
groups = data.get("groups", {}) or {}
if groups:
    lines.append("== GROUPS & ORGANISATIONS ==")
    for name, entry in groups.items():
        if not isinstance(entry, dict):
            lines.append(f"- {name}")
            continue
        desc = (entry.get("description") or "").strip().replace("\n", " ")
        if desc:
            lines.append(f"- {name}: {desc}")
        else:
            lines.append(f"- {name}")
```

```
        lines.append("")

    # Locations
    locations = data.get("locations", {}) or {}
    if locations:
        lines.append("== LOCATIONS ==")
        for name, entry in locations.items():
            if not isinstance(entry, dict):
                lines.append(f"- {name}")
                continue
            desc = (entry.get("description") or "").strip().replace("\n", " ")
            sig = (entry.get("significance") or "").strip()
            detail = " — ".join(filter(None, [desc, sig]))
            if detail:
                lines.append(f"- {name}: {detail}")
            else:
                lines.append(f"- {name}")
        lines.append("")

    # Terms (descriptions only, no aliases)
    terms = data.get("terms", {}) or {}
    if terms:
        lines.append("== TERMS ==")
        for name, entry in terms.items():
            if not isinstance(entry, dict):
                lines.append(f"- {name}")
                continue
            desc = (entry.get("description") or "").strip().replace("\n", " ")
            if desc:
                lines.append(f"- {name}: {desc}")
            else:
                lines.append(f"- {name}")
        lines.append("")

    return "\n".join(lines)


def main():
    script_dir = Path(__file__).resolve().parent
    meta_dir = (script_dir / ".." / "meta").resolve()
```

```
    parser = argparse.ArgumentParser(
        description="Generate a filtered summary context from a YAML meta file."
    )
    parser.add_argument("--game", required=True, metavar="SLUG",
                        help="Game slug — resolves to meta/<slug>.yaml")
    args = parser.parse_args()

    yaml_path = meta_dir / f"{args.game}.yaml"
    if not yaml_path.exists():
        print(f"□ YAML not found: {yaml_path}", file=sys.stderr)
        sys.exit(1)

    data = load_yaml(yaml_path)
    print(build_context(data))


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