# The Scripts

TranscriptOMatic consists of the following scripts.

In addition: Don't forget to create the needed files and folders and set the right permissions.

- Scripts in meetings/lib/
- Script meetings/bin/meeting-start
- Script meetings/bin/meeting-stop
- Script meetings/bin/meeting-follow

# Scripts in meetings/lib/

**Scripts used by the executable scripts, providing basic configuration information.**

## **Script**: `meetings/lib/paths.sh`

Creates:

- Session Directory
- Session Files
  - `audio.wav`
  - `transcript.txt`
  - `meta.env`
- Timestamps (ISO)

and

- Provides path information for all scripts

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

BASE="$HOME/meetings/recordings"
mkdir -p "$BASE"

# ISO-like stamp with seconds
ISO_STAMP="$(date +%Y-%m-%dT%H%M%S)"

SESSION="$BASE/$ISO_STAMP"
i=0
while [[ -e "$SESSION" ]]; do
  i=$((i + 1))
  SESSION="$BASE/${ISO_STAMP}_$i"
done

mkdir -p "$SESSION"

AUDIO="$SESSION/audio.wav"
TRANSCRIPT="$SESSION/transcript.txt"
```

```
META="$SESSION/meta.env"
CURRENT="$BASE/.current"
```

## **Script**: `meetings/lib/whisper.sh`

Contains:

- the path to the local `whisper.cpp` installation
- the language model used (currently `ggml-tiny.en.bin`)
- ASR (Automatic Speech Recognition) parameter used

```
#!/usr/bin/env bash

# whisper.cpp streaming binary
WHISPER="$HOME/whisper.cpp/build/bin/whisper-stream"

# Models
MODEL_EN="$HOME/whisper.cpp/models/ggml-tiny.en.bin"
MODEL_MULTI="$HOME/whisper.cpp/models/ggml-base.bin"

# Threading (Raspberry Pi 500 defaults)
INFER_THREADS=4
MEL_THREADS=16

# whisper.sh — Streaming / context parameters
STREAM_OPTS=(
  -c 1
  --keep-context
  --step 8000
  --length 12000
  --keep 800
)
# Thread options
THREAD_OPTS=(
  -t "$INFER_THREADS"
  -mt "$MEL_THREADS"
)
```

# Script meetings/bin/meeting-start

**Starts a new live transcription session.**

`./bin/meeting-start`

## Usage

- ~~German:~~ `~~meeting-start --de~~`
- English: `meeting-start --en`
- ~~Auto:~~ `~~meeting-start --auto~~`

> --de and --auto do NOT work at this stage of the project. Only use --en.

```
meeting-start --en
```

## Script

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

HERE="$(cd "$(dirname "$0")" && pwd)"
source "$HERE/../lib/paths.sh"
source "$HERE/../lib/whisper.sh"

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

LANG_MODE="en"
NO_AUDIO=0
NO_TRANSCRIPT=0
```

```
while [[ $# -gt 0 ]]; do
  case "$1" in
    --de)            LANG_MODE="de";    shift ;;
    --en)            LANG_MODE="en";    shift ;;
    --auto)          LANG_MODE="auto";  shift ;;
    --no-audio)      NO_AUDIO=1;        shift ;;
    --no-transcript) NO_TRANSCRIPT=1;   shift ;;
    *)
      echo "Usage: meeting-start [--de|--en|--auto] [--no-audio] [--no-transcript]"
      exit 2
      ;;
  esac
done

if [[ "$NO_AUDIO" -eq 1 && "$NO_TRANSCRIPT" -eq 1 ]]; then
  echo "□ --no-audio and --no-transcript together: nothing to do." >&2
  exit 2
fi

# ------------------------------------------------------------
# Select model + language options for whisper-stream
# ------------------------------------------------------------

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

# ------------------------------------------------------------
# Audio setup: persistent sink + virtual microphone
```

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

DISCORD_SINK="discord_sink"
DISCORD_MONITOR="${DISCORD_SINK}.monitor"
VIRTUAL_MIC="whisper_mic"

echo "□□Session: $SESSION"
echo "□□□ Language mode: $LANG_MODE"
echo "□□□ Audio recording: $([ "$NO_AUDIO" -eq 1 ] && echo 'off (--no-audio)' || echo 'on')"
echo "□□Live transcript: $([ "$NO_TRANSCRIPT" -eq 1 ] && echo 'off (--no-transcript)' || echo
'on')"
echo "□□Setting up audio routing…"
echo "   Sink:   $DISCORD_SINK"
echo "   Source: $VIRTUAL_MIC (master: $DISCORD_MONITOR)"
echo "----"

# 1) Ensure discord_sink exists
if ! pactl list short sinks | awk '{print $2}' | grep -qx "$DISCORD_SINK"; then
  pactl load-module module-null-sink \
    sink_name="$DISCORD_SINK" \
    sink_properties=device.description=DiscordSink >/dev/null
fi

# 2) Try to move an active sink-input (Discord) to discord_sink
#    This may be transient in silent channels → poll briefly.
moved="no"
for _ in {1..40}; do
  SID="$(pactl list short sink-inputs 2>/dev/null | awk 'NF{print $1}' | head -n1 || true)"
  if [[ -n "${SID:-}" ]]; then
    if pactl move-sink-input "$SID" "$DISCORD_SINK" 2>/dev/null; then
      moved="yes"
      break
    fi
  fi
  sleep 0.25
done

if [[ "$moved" != "yes" ]]; then
  echo "⚠□  Could not move a sink-input to $DISCORD_SINK."
  echo "    Make sure Legcord is connected to a voice channel, then re-run meeting-start."
```

```
fi

# 3) Remove existing whisper_mic remap-sources (idempotent)
for mid in $(pactl list short modules | awk '$0 ~ /module-remap-source/ && $0 ~
/source_name=whisper_mic/ {print $1}'); do
  pactl unload-module "$mid" >/dev/null 2>&1 || true
done

# 4) Create virtual microphone (mono; attempt 16 kHz)
REMAPPED_MID="$(pactl load-module module-remap-source \
  master="$DISCORD_MONITOR" \
  source_name="$VIRTUAL_MIC" \
  channels=1 \
  rate=16000 \
  master_channel_map=front-left \
  channel_map=mono \
  source_properties=device.description=WhisperMic16k)"

echo "□ Audio ready (remap module id: $REMAPPED_MID)"
echo "----"

# ------------------------------------------------------------
# Start recording (ffmpeg) and/or transcription (whisper-stream)
# ------------------------------------------------------------

FFMPEG_PID=""
WHISPER_PID=""

# --- ffmpeg: record to WAV ---
if [[ "$NO_AUDIO" -eq 0 ]]; then
  if [[ -e "$AUDIO" ]]; then
    echo "□ Refusing to overwrite existing file: $AUDIO" >&2
    echo "   (Session dir collision or leftover file)" >&2
    exit 1
  fi

  ffmpeg -nostdin -y -hide_banner -loglevel error \
    -f pulse -i "$VIRTUAL_MIC" \
    -ac 1 -ar 16000 \
    "$AUDIO" &
```

```
  FFMPEG_PID=$!
  echo "□□□ Recording to: $AUDIO"
fi

# --- whisper-stream: live transcription ---
# whisper-stream reads directly from the capture device (-c 1 in STREAM_OPTS).
# Do NOT pass -f: for whisper-stream, -f is a text output file, not an audio input.
if [[ "$NO_TRANSCRIPT" -eq 0 ]]; then
  "$WHISPER" \
    -m "$MODEL" \
    "${LANG_OPTS[@]}" \
    "${STREAM_OPTS[@]}" \
    "${THREAD_OPTS[@]}" \
    | tee -a "$TRANSCRIPT" &
  WHISPER_PID=$!
  echo "□□Live transcription to: $TRANSCRIPT"
fi

echo "→□  Follow the transcript with: meeting-follow"
echo "□  Stop with: meeting-stop"
echo "----"

# Announce active session
echo "$SESSION" > "$CURRENT"

# ------------------------------------------------------------
# Metadata
# ------------------------------------------------------------

cat > "$META" <<EOF
SESSION=$SESSION
AUDIO=${AUDIO:-}
TRANSCRIPT=${TRANSCRIPT:-}
LANG_MODE=$LANG_MODE
MODEL=$MODEL
NO_AUDIO=$NO_AUDIO
NO_TRANSCRIPT=$NO_TRANSCRIPT
DISCORD_SINK=$DISCORD_SINK
DISCORD_MONITOR=$DISCORD_MONITOR
VIRTUAL_MIC=$VIRTUAL_MIC
```

```
REMAPPED_MID=$REMAPPED_MID
FFMPEG_PID=${FFMPEG_PID:-}
WHISPER_PID=${WHISPER_PID:-}
EOF

[[ -n "${FFMPEG_PID:-}" ]]  && echo "$FFMPEG_PID"  > "$SESSION/ffmpeg.pid"
[[ -n "${WHISPER_PID:-}" ]] && echo "$WHISPER_PID" > "$SESSION/whisper.pid"

echo "□ meeting-start finished setup successfully."
```

# Script meetings/bin/meeting-stop

`./bin/meeting-stop`

**Stops the active transcription session and finalises all session files.**

## Usage

```
meeting-stop
```

## Script

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

BASE="$HOME/meetings/recordings"
CURRENT="$BASE/.current"

SESSION="${1:-$(ls -d "$HOME/meetings/recordings/"* | grep -v '^\.' | tail -n1)}"
META="$SESSION/meta.env"

# ------------------------------------------------------------
# Stop running processes
# ------------------------------------------------------------

if [[ -f "$SESSION/ffmpeg.pid" ]]; then
  kill "$(cat "$SESSION/ffmpeg.pid")" 2>/dev/null || true
fi

if [[ -f "$SESSION/whisper.pid" ]]; then
  kill "$(cat "$SESSION/whisper.pid")" 2>/dev/null || true
fi

# Remove current session pointer
rm -f "$CURRENT"
```

```
# ------------------------------------------------------------
# Load metadata (if available)
# ------------------------------------------------------------

if [[ -f "$META" ]]; then
  # shellcheck disable=SC1090
  source "$META"
fi

# ------------------------------------------------------------
# Clean up audio graph (optional but recommended)
# ------------------------------------------------------------

if [[ -n "${REMAPPED_MID:-}" ]]; then
  pactl unload-module "$REMAPPED_MID" >/dev/null 2>&1 || true
fi

# ------------------------------------------------------------
# Ask for meeting name
# ------------------------------------------------------------

read -r -p "Meeting name (for filenames): " MEETING_NAME
MEETING_NAME="${MEETING_NAME:-meeting}"

# Slugify: lowercase, umlauts, spaces → _
slugify() {
  local s="$1"
  s="$(printf '%s' "$s" | tr '[:upper:]' '[:lower:]')"
  s="$(printf '%s' "$s" | sed \
    -e 's/ä/ae/g' -e 's/ö/oe/g' -e 's/ü/ue/g' -e 's/ß/ss/g' \
    -e 's/[^a-z0-9 _-]/_/g' \
    -e 's/[[:space:]]\+/_/g' \
    -e 's/_\{2,\}/_/g' \
    -e 's/^_//' -e 's/_$//')"
  printf '%s' "$s"
}

SLUG="$(slugify "$MEETING_NAME")"
```

```
# ------------------------------------------------------------
# Final filenames
# ------------------------------------------------------------

ISO="${ISO_STAMP:-$(date +%Y-%m-%dT%H%M)}"

NEW_TRANSCRIPT=""
NEW_AUDIO=""

if [[ -f "$SESSION/transcript.txt" ]]; then
  NEW_TRANSCRIPT="$SESSION/${ISO}_${SLUG}_transcript.txt"
  mv -n "$SESSION/transcript.txt" "$NEW_TRANSCRIPT"
fi

if [[ -f "$SESSION/audio.wav" ]]; then
  NEW_AUDIO="$SESSION/${ISO}_${SLUG}_audio.wav"
  mv -n "$SESSION/audio.wav" "$NEW_AUDIO"
fi

# ------------------------------------------------------------
# Final output
# ------------------------------------------------------------

echo "□ Meeting stopped."
echo "□□Session: $SESSION"
[[ -n "$NEW_AUDIO" ]]      && echo "□□Audio:      $NEW_AUDIO"
[[ -n "$NEW_TRANSCRIPT" ]] && echo "□□Transcript: $NEW_TRANSCRIPT"
echo "□□Meta: $META"
```

# Script meetings/bin/meeting-follow

`./bin/meeting-follow`

**Display the transcript while it is generated.**

## Usage

```
meeting-follow
```

## Script

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

BASE="$HOME/meetings/recordings"
CURRENT="$BASE/.current"

echo "□□Waiting for active meeting session…"

# Wait until a current session is announced
while [[ ! -f "$CURRENT" ]]; do
  sleep 0.5
done

SESSION="$(cat "$CURRENT")"
META="$SESSION/meta.env"
TRANSCRIPT="$SESSION/transcript.txt"

# Check if live transcription is disabled for this session
if [[ -f "$META" ]]; then
  # shellcheck disable=SC1090
  source "$META"
```

```
fi

if [[ "${NO_TRANSCRIPT:-0}" -eq 1 ]]; then
  echo "⚠  This session was started with --no-transcript."
  echo "   No live transcript available."
  exit 1
fi

echo "Following transcript:"
echo "   Session: $SESSION"
echo "   File:    $TRANSCRIPT"
echo "----"

# Wait until transcript file exists
while [[ ! -f "$TRANSCRIPT" ]]; do
  sleep 0.5
done

tail -f "$TRANSCRIPT"
```

Expected behaviour:

`meeting-follow` will check the file `$HOME/meetings/recordings/.current` if an active transcript session exists. If not, it waits until the information about the active session has been added by `meeting-start` and will then start displaying the transcript.