Creating (Almost) Live Transcripts

From Voice to Text

```
Discord (Legcord)
   ↓
PipeWire graph
   ↓
discord_sink (virtual null sink)
   ↓
discord_sink.monitor (loopback source)
   ↓
whisper_mic (remap-source, mono, 16kHz)
   ↓
ffmpeg
   ↓
audio.wav (growing file)
   ↓
whisper-stream
```

On the device used for transcription, a Discord client (Legcord) is running and joins the session's Discord voice channel. Legcord's audio output is moved to a virtual null sink (named discord_sink) via PipeWire. The sink's loopback source (discord_sink.monitor) is remapped by whisper_mic and fed into ffmpeg.

Whisper-stream uses the wav file created by ffmpeg to create the transcript.

Using a minimal language model (tiny.en), the voice input is transcribed with a delay of 5 to 10 seconds.

Structure

```
meetings/
├── bin/
│   ├── meeting-start          # starts recording + live transcription
│   ├── meeting-stop           # stops recording, asks for meeting name, renames files
│   ├── meeting-follow         # follow transcript while it is being written
│   └── summarize-meeting      # create post-meeting summaries (planned, not yet realized)
│
├── lib/
│   ├── paths.sh               # creates session dirs + defines file paths
│   └── whisper.sh             # whisper.cpp binary, model, ASR parameters
│
└── recordings/
    └── 2026-02-21T013852/     # session directory (created on meeting-start)
        ├── audio.wav          # raw system audio recording
        ├── transcript.txt     # live transcript (grows during meeting)
        ├── meta.env           # session metadata (PIDs, language, timestamps)
        ├── 2025-03-24T1930_project-sync_transcript.txt   # the actual renamed transcript
        ├── 2025-03-24T1930_project-sync_audio.wav        # (see data protection discussion)
        └── summary.md         # created by summarize-meeting (planned, not yet realized)
```

Description of Paths and Scripts

Paths

Path: `meetings/bin/`

- Executable scripts.

Path: `meetings/lib/`

- Scripts to be used by the executable scripts.

Path: `meetings/recordings/`

- Subdirectories (ISO timestamp format) containing:
  - Audio files (should be deleted after the transcript has been written)
  - `transcript.txt` (renamed and timestamped after the meeting's end by the `meeting-stop` script)
  - `meta.env` (Meeting information)
    - PIDs
    - language used
    - the meeting's timestamps
  - `summary.md` (meeting summary in Markdown format)

Scripts (current architecture)

TranscriptOMatic is implemented as a small set of composable shell scripts. Each script has a clearly defined responsibility within the session lifecycle. No script relies on implicit system state or hard-coded audio devices.

Library scripts (meetings/lib/)

`meetings/lib/paths.sh`

Responsible for **session creation and path management**.

On invocation, it:

- creates a new session directory

```
~/meetings/recordings/<ISO_TIMESTAMP>/
```

- defines canonical file locations:
  - `audio.wav`
  - `transcript.txt`
  - `meta.env`
- provides these paths to all other scripts

This script is the *only* place where session directories are created.

`meetings/lib/whisper.sh`

Defines the **speech recognition backend configuration**.

It contains:

- the path to the local `whisper.cpp` installation
- model selection (language-specific vs. multilingual)
- streaming parameters
- threading configuration suitable for a Raspberry Pi–class system

The setup is explicitly optimized for **live transcription** using `whisper-stream`, not for batch processing.

No audio devices are referenced here.

Executable scripts (meetings/bin/)

meeting-start

Starts a new live transcription session.

Responsibilities

1. **Session initialization**
   - creates a new session directory via `paths.sh`
   - writes the active session path to `~/meetings/recordings/.current`
2. **Audio graph setup (PipeWire)**
   - ensures a persistent null sink (`discord_sink`)
   - routes Discord audio into that sink
   - exposes the sink monitor as a virtual microphone (`whisper_mic`)
3. **Processing**
   - records audio from `whisper_mic` via `ffmpeg`
   - performs (almost) live transcription using `whisper-stream`
   - appends output to `transcript.txt`
4. **State tracking**
   - writes all relevant runtime information (PIDs, module IDs, paths) to `meta.env`

Usage

```
meeting-start --en    # force English
```

Planned, not yet realized:

```
meeting-start --de    # force German (planned, not yet realized)
meeting-start --auto  # auto-detect language (planned, not yet realized)
```

`meeting-start` is self-contained: it does not require any pre-existing audio configuration and can be run after a reboot.

meeting-follow

Passively follows the live transcript of the **currently active session**.

Behaviour

- waits for the presence of `~/meetings/recordings/.current`
- reads the active session path from that file
- waits until `transcript.txt` exists
- follows the transcript in real time

This allows meeting-follow to be started:

- before meeting-start
- over SSH
- in a shared terminal window

It will never attach to archived sessions.

Usage

```
meeting-follow # stop with ctrl+c
```

meeting-stop

Stops the active transcription session and finalises all session files.

Responsibilities

1. **Process teardown** kills the FFmpeg and whisper-stream processes via their stored PIDs
2. **State clean-up** removes `~/meetings/recordings/.current` unloads the `whisper_mic` remap-source module from PipeWire
3. **File finalisation** prompts for a meeting name and slugifies it
   - renames transcript.txt and audio.wav to timestamped, named files (e.g. 2025-03-24T1930_project-sync_transcript.txt)

To prevent accidental deletions, `meeting-stop` does not delete files automatically. To maintain data protection, recording files have to be deleted by hand.

Usage

```
meeting-stop             # stops the most recent session
```