Nice — I’ll give you a complete, ready-to-run project package: full file tree, every source file (with the **fast Vosk streaming** offline behavior, Whisper fallback, Google online-only, commit-by-file support, robust logging with immediate flush), step-by-step installation + Git/GitHub connection instructions, and Streamlit dashboard that shows logs in near real-time.

Per your request I **did not** change two things you told me not to change:

* I did **not** add a special if "\_\_PUSH\_\_" in cmds: ... check in the main loop.
* I left push\_current\_branch() in the same shape/functionality you specified (it’s present and used elsewhere).

Below are the files and the detailed steps. Copy the files into your project folder and follow the installation steps.

**Project structure**

voice\_git\_automation\_final/

│

├─ main.py # main voice controller (full code below)

├─ dashboard.py # Streamlit dashboard (real-time logs)

├─ requirements.txt # pip dependencies

├─ .gitignore

├─ logs/

│ └─ commands.txt # auto created logs (ignored by git)

├─ command\_log.txt # short command log (ignored by git)

├─ auth/

│ └─ voice\_ref.wav # optional voice reference for auth

├─ models/

│ └─ vosk-model-small-en-us-0.15/ # put Vosk model here

└─ README.md # instructions (below)

**1) main.py — full source (copy this file exactly)**

# main.py

"""

Voice-Guided Git Automation (single-file)

- Vosk (offline streaming) + Google (online) + Whisper fallback

- Fast offline (streaming) recognition, Whisper fallback if Vosk empty

- Voice auth optional

- NLP parser supports "commit file <name>" and commit messages

- Logs written and flushed for real-time dashboard

- Auto-push after commit is attempted inside execute\_git\_sequence (if network)

NOTE: I DID NOT add a special "\_\_PUSH\_\_" main-loop check and did not modify your

push\_current\_branch() function shape — per your request.

"""

import os

import sys

import time

import re

import json

import wave

import subprocess

import threading

import datetime

import queue

import requests

import numpy as np

# ASR libraries

try:

from vosk import Model, KaldiRecognizer

except Exception:

Model = None

import pyaudio

import sounddevice as sd

import speech\_recognition as sr

# Whisper (fallback)

try:

import whisper

except Exception:

whisper = None

# TTS & audio processing

import pyttsx3

import librosa

# NLP (spaCy optional)

try:

import spacy

nlp = spacy.load("en\_core\_web\_sm")

except Exception:

nlp = None

# ---------------------- CONFIG ----------------------

SAMPLE\_RATE = 16000

VOSK\_MODEL\_PATH = os.path.join(os.getcwd(), "models", "vosk-model-small-en-us-0.15")

WHISPER\_MODEL = "tiny" # tiny | base | small ; tiny is fastest

VOICE\_AUTH\_REF = os.path.join("auth", "voice\_ref.wav")

LOGS\_DIR = "logs"

COMMAND\_LOG = "command\_log.txt"

ENABLE\_VOICE\_AUTH = False # set True to require voice auth for push/commit

# ----------------------------------------------------

# ensure logs dir exists

os.makedirs(LOGS\_DIR, exist\_ok=True)

# TTS

engine = pyttsx3.init()

def speak(text):

# keep TTS minimal to avoid blocking too long; print as well

print("🗣️", text)

try:

engine.say(text)

engine.runAndWait()

except Exception:

pass

# ---------------------- LOGGING (flush immediately) ----------------------

def append\_log(line):

"""Write the same timestamped line to logs/commands.txt and short command\_log.txt (flushed)."""

ts = datetime.datetime.now().isoformat(sep=' ', timespec='seconds')

log\_line = f"{ts} - {line}\n"

# detailed log

dpath = os.path.join(LOGS\_DIR, "commands.txt")

with open(dpath, "a", encoding="utf-8") as f:

f.write(log\_line)

f.flush()

try:

os.fsync(f.fileno())

except Exception:

pass

# short command log (also flushed)

with open(COMMAND\_LOG, "a", encoding="utf-8") as f2:

f2.write(log\_line)

f2.flush()

try:

os.fsync(f2.fileno())

except Exception:

pass

# ---------------------- ASR INIT ----------------------

vosk\_model = None

if Model is not None and os.path.exists(VOSK\_MODEL\_PATH):

try:

vosk\_model = Model(VOSK\_MODEL\_PATH)

print("✅ Vosk model loaded from", VOSK\_MODEL\_PATH)

except Exception as e:

print("⚠️ Failed to load Vosk model:", e)

whisper\_model = None

if whisper is not None:

try:

whisper\_model = whisper.load\_model(WHISPER\_MODEL)

print(f"✅ Whisper model '{WHISPER\_MODEL}' loaded.")

except Exception as e:

print("⚠️ Whisper load failed:", e)

# ---------------------- UTIL: record small snippet (silent) ----------------------

def record\_wav(filename="fallback.wav", duration=3, fs=SAMPLE\_RATE):

"""Record short snippet quickly (used only for Whisper fallback or voice auth)."""

# Do not speak here (keeps fallback fast)

rec = sd.rec(int(duration \* fs), samplerate=fs, channels=1, dtype='int16')

sd.wait()

with wave.open(filename, "wb") as wf:

wf.setnchannels(1)

wf.setsampwidth(2)

wf.setframerate(fs)

wf.writeframes(rec.tobytes())

return filename

# ---------------------- OFFLINE ASR (fast Vosk streaming via sounddevice) ----------------------

\_audio\_queue = queue.Queue()

def \_sd\_callback(indata, frames, time\_info, status):

if status:

# print status but don't flood

print("SoundDevice status:", status)

# push bytes to queue

\_audio\_queue.put(bytes(indata))

def offline\_asr\_vosk(timeout=5):

"""

Fast streaming Vosk listener. Returns recognized text or ''.

Uses sounddevice.RawInputStream and a queue for real-time audio.

"""

if vosk\_model is None:

return ""

rec = KaldiRecognizer(vosk\_model, SAMPLE\_RATE)

result\_text = ""

# raw stream ensures we get 16-bit ints suitable for Vosk

try:

with sd.RawInputStream(samplerate=SAMPLE\_RATE, blocksize=8000,

dtype='int16', channels=1, callback=\_sd\_callback):

print("🗣️ Listening (offline)... speak now.")

start = time.time()

while True:

# timeout guard

if time.time() - start > timeout:

# try final partial

try:

final = json.loads(rec.FinalResult())

result\_text = final.get("text", "")

except Exception:

result\_text = ""

break

try:

data = \_audio\_queue.get(timeout=timeout)

except queue.Empty:

continue

if rec.AcceptWaveform(data):

res = json.loads(rec.Result())

result\_text = res.get("text", "").strip()

break

# else keep looping

except Exception as e:

print("Vosk streaming error:", e)

result\_text = ""

return (result\_text or "").strip()

# ---------------------- CLOUD (Google) ASR ----------------------

def cloud\_asr\_google(timeout=6):

r = sr.Recognizer()

with sr.Microphone(sample\_rate=SAMPLE\_RATE) as source:

r.adjust\_for\_ambient\_noise(source, duration=0.5)

speak("Listening (online)... speak now.")

try:

audio = r.listen(source, timeout=timeout, phrase\_time\_limit=8)

except Exception as e:

print("Listen error:", e)

return ""

try:

text = r.recognize\_google(audio)

return text.strip()

except sr.UnknownValueError:

return ""

except sr.RequestError as e:

print("Google API error:", e)

return ""

# ---------------------- WHISPER fallback ----------------------

def whisper\_transcribe\_from\_file(filename="fallback.wav"):

if whisper\_model is None:

return ""

try:

res = whisper\_model.transcribe(filename)

return res.get("text", "").strip()

except Exception as e:

print("Whisper error:", e)

return ""

# ---------------------- HYBRID ASR ----------------------

def hybrid\_listen(mode="offline"):

"""Return recognized text (lowercase) or ''."""

text = ""

if mode == "offline":

# Vosk first (fast)

text = offline\_asr\_vosk(timeout=5)

if not text:

# fallback → Whisper (silent short record)

record\_wav("fallback.wav", duration=3)

text = whisper\_transcribe\_from\_file("fallback.wav")

elif mode == "online":

# Google only (no fallback)

text = cloud\_asr\_google(timeout=6)

elif mode == "whisper":

# Whisper only

record\_wav("fallback.wav", duration=3)

text = whisper\_transcribe\_from\_file("fallback.wav")

return (text or "").lower().strip()

# ---------------------- VOICE AUTH (simple MFCC) ----------------------

def extract\_mfcc(file):

y, sr\_ = librosa.load(file, sr=SAMPLE\_RATE)

mf = np.mean(librosa.feature.mfcc(y=y, sr=sr\_, n\_mfcc=13), axis=1)

return mf

def authenticate\_via\_voice(timeout=3, threshold=55.0):

if not os.path.exists(VOICE\_AUTH\_REF):

print("Voice reference not found; skipping voice auth.")

return False

path = record\_wav("auth\_tmp.wav", duration=timeout)

try:

ref = extract\_mfcc(VOICE\_AUTH\_REF)

usr = extract\_mfcc(path)

dist = np.linalg.norm(ref - usr)

print("Voice distance:", dist)

return dist < threshold

except Exception as e:

print("Voice auth error:", e)

return False

# ---------------------- NLP -> git command parser ----------------------

def parse\_git\_command(text):

text = text.lower().strip()

# --- Commit with custom message (all files) ---

m = re.search(r"commit (?:message )?[\"'](.+?)[\"']", text)

if "commit" in text and m and "with file" not in text:

message = m.group(1)

return ["git add -A", f'git commit -m "{message}"']

# --- Commit specific file(s) ---

# Example: "commit dashboard.py" OR "commit main.py with message 'fix bug'"

m\_file = re.search(r"commit ([\w\.\-/]+)(?: with message [\"'](.+?)[\"'])?", text)

if m\_file:

filename = m\_file.group(1)

message = m\_file.group(2) if m\_file.group(2) else f"voice commit {filename}"

return [f"git add {filename}", f'git commit -m "{message}"']

# --- Commit all (default) ---

if "commit all" in text or text.strip() == "commit":

return ["git add -A", 'git commit -m "voice commit"']

# --- Push ---

if "push" in text:

# leave actual current-branch push to push\_current\_branch()

return ["git push"]

# --- Pull ---

if "pull" in text:

return ["git pull"]

# --- Status ---

if "status" in text:

return ["git status"]

# --- Create branch ---

if "create branch" in text:

b = re.search(r"create branch ([\w\-\_]+)", text)

if b:

bn = b.group(1)

return [f"git branch {bn}", f"git checkout {bn}"]

return ["Unknown command"]

# --- Switch branch ---

if "switch to" in text or "checkout" in text:

b = re.search(r"(?:switch to|checkout) (?:branch )?([\w\-\_]+)", text)

if b:

return [f"git checkout {b.group(1)}"]

# --- Undo last commit ---

if "undo last commit" in text or "revert last commit" in text or "undo" in text:

return ["git reset --soft HEAD~1"]

return ["Unknown command"]

# ---------------------- GIT HELPERS ----------------------

def is\_git\_repo():

res = subprocess.run(["git", "rev-parse", "--is-inside-work-tree"], capture\_output=True, text=True)

return res.returncode == 0

def current\_branch():

r = subprocess.run(["git", "rev-parse", "--abbrev-ref", "HEAD"], capture\_output=True, text=True)

if r.returncode == 0:

return r.stdout.strip()

return "main"

def run\_cmd(cmd):

"""Run shell command list or string; return (rc, stdout+stderr)"""

if isinstance(cmd, str):

shell = True

else:

shell = False

print("⚡ Running:", cmd)

res = subprocess.run(cmd, shell=shell, capture\_output=True, text=True)

out = (res.stdout or "") + (res.stderr or "")

print(out)

return res.returncode, out

def execute\_git\_sequence(cmds):

"""Execute a sequence of git shell commands (strings)."""

if not is\_git\_repo():

speak("Not a Git repository. Please run 'git init' or set the correct folder.")

return False

for c in cmds:

rc, out = run\_cmd(c)

if rc != 0:

speak(f"Command failed: {c.split()[0]}")

append\_log(f"FAILED: {c} -> {out}")

return False

# If we just committed, try to push to remote (attempt; will fail gracefully if offline)

if any("commit" in c for c in cmds):

try:

# quick network test

requests.get("https://github.com", timeout=3)

# if reachable, push current branch

push\_current\_branch()

except Exception as e:

append\_log(f"Auto-push skipped (offline): {e}")

print("⚠️ Auto-push skipped (offline).")

return True

def push\_current\_branch():

br = current\_branch()

# use safe push: origin <branch>

rc, out = run\_cmd(["git", "push", "origin", br])

if rc == 0:

speak(f"Pushed to origin {br}")

append\_log(f"push -> origin/{br}")

return True

else:

speak("Push failed. Check credentials or remote.")

append\_log(f"PUSH FAILED -> origin/{br} : {out}")

return False

# ---------------------- MAIN LOOP ----------------------

def main():

speak("Voice Git Automation Started. Please say 'offline' or 'online' to choose ASR mode.")

# initial selection

mode = "offline"

attempts = 0

while attempts < 4:

text = hybrid\_listen(mode="online") # quick online selection

if not text:

attempts += 1

continue

if "offline" in text:

mode = "offline"

speak("Selected offline mode.")

break

if "online" in text:

mode = "online"

speak("Selected online mode.")

break

attempts += 1

speak("Please say 'offline' or 'online' to select ASR mode.")

speak(f"Using {mode} ASR mode. Say a git command, or say 'exit' to stop.")

try:

while True:

text = hybrid\_listen(mode=mode)

if not text:

continue

append\_log(text)

print("Recognized:", text)

# mode switching voice commands

if "switch to offline" in text:

mode = "offline"

speak("Switched to offline mode.")

continue

if "switch to online" in text:

mode = "online"

speak("Switched to online mode.")

continue

if text.strip() in ("exit", "quit", "stop"):

speak("Exiting. Goodbye.")

break

cmds = parse\_git\_command(text)

if cmds[0] == "Unknown command":

speak("Unknown Git command. Try: commit, push, pull, status, create branch, switch to ...")

continue

# require voice auth for sensitive ops if enabled

if ENABLE\_VOICE\_AUTH and any(k in " ".join(cmds) for k in ("commit", "push", "reset", "branch")):

speak("Please authenticate with voice.")

ok = authenticate\_via\_voice(timeout=3)

if not ok:

speak("Authentication failed.")

append\_log("AUTH FAIL for command: " + " | ".join(cmds))

continue

speak("Authentication passed.")

# fast immediate execution

speak(f"Executing {', '.join(cmds)}")

ok = execute\_git\_sequence(cmds)

if ok and any("push" in c for c in cmds):

# if user said push, do push\_current\_branch() to ensure origin/branch

push\_current\_branch()

append\_log("EXECUTED: " + " | ".join(cmds))

except KeyboardInterrupt:

speak("Stopped by keyboard.")

except Exception as e:

print("Main loop error:", e)

finally:

try:

engine.stop()

except Exception:

pass

if \_\_name\_\_ == "\_\_main\_\_":

main()

**2) dashboard.py — Streamlit live log viewer (copy this file)**

# dashboard.py

import streamlit as st

import os

import time

COMMAND\_LOG\_PATH = "command\_log.txt"

LOGS\_DIR = "logs"

LOGS\_FILE\_PATH = os.path.join(LOGS\_DIR, "commands.txt")

st.set\_page\_config(page\_title="Voice-Guided Git Automation", layout="wide")

st.title("🗣️ Voice-Guided Git Automation Dashboard")

st.markdown("### ✅ Real-time log view of your Git voice commands")

def load\_log\_file(path):

if os.path.exists(path):

try:

with open(path, "r", encoding="utf-8") as f:

return f.read()

except Exception as e:

return f"Error reading log: {e}"

return "⚠️ No logs available yet."

# Create persistent containers once

command\_container = st.empty()

detail\_container = st.empty()

# header (fixed)

command\_container.subheader("📌 Command Execution Log (command\_log.txt)")

detail\_container.subheader("📝 Detailed Logs (logs/commands.txt)")

# text areas stored in their own placeholders

cmd\_area = st.empty()

det\_area = st.empty()

REFRESH\_SECONDS = 1

while True:

cmd\_text = load\_log\_file(COMMAND\_LOG\_PATH)

det\_text = load\_log\_file(LOGS\_FILE\_PATH)

# update areas

cmd\_area.text\_area("Command Log (auto-refresh)", cmd\_text, height=300, key="cmd\_log\_area")

det\_area.text\_area("Detailed Log (auto-refresh)", det\_text, height=300, key="det\_log\_area")

time.sleep(REFRESH\_SECONDS)

# Streamlit will re-run; loop keeps updating

**3) .gitignore (create in repo root)**

# Ignore logs and fallback audio

logs/

command\_log.txt

\*.wav

\_\_pycache\_\_/

\*.pyc

.env

.vscode/

venv/

**4) requirements.txt**

sounddevice>=0.4.6

pyaudio>=0.2.11

vosk>=0.3.45

whisper @ git+https://github.com/openai/whisper.git@master

speechrecognition>=3.8.1

pyttsx3>=2.90

numpy>=1.23

librosa>=0.9.2

streamlit>=1.20

spacy>=3.5

requests>=2.28

Note: whisper line installs from GitHub (OpenAI whisper Python repo). Alternatively use openai-whisper package name if available in your environment.

**5) README / Step-by-step install & run (detailed)**

**A. Prerequisites**

* Python 3.9+ (3.10/3.11 fine)
* Git installed
* On Windows: install pipwin then pipwin install pyaudio if pip install pyaudio fails.
* Enough disk space for Vosk model (~50–200MB for small models).
* (Optional) GPU for Whisper (if you plan to use larger Whisper models). Otherwise tiny/base ok on CPU but slower.

**B. Setup (Windows / macOS / Linux)**

1. Clone repo (or place files):
2. git clone https://github.com/khushi-2023/voice\_git\_automation\_final.git
3. cd voice\_git\_automation\_final
4. Create and activate a virtual environment:
5. python -m venv venv
6. # Windows
7. venv\Scripts\activate
8. # macOS/Linux
9. source venv/bin/activate
10. Install dependencies:
11. pip install --upgrade pip
12. pip install -r requirements.txt
13. Install spaCy language model:
14. python -m spacy download en\_core\_web\_sm
15. Download Vosk model:
    * Visit <https://alphacephei.com/vosk/models>
    * Download vosk-model-small-en-us-0.15 (recommended small model)
    * Unzip and ensure folder is models/vosk-model-small-en-us-0.15 relative to project root.
16. (Optional) Put a voice reference file for voice auth:
    * auth/voice\_ref.wav (16kHz mono). If not present, voice auth will be skipped.

**C. GitHub authentication (so voice-triggered push works)**

You must configure Git to allow push without interactive username/password prompts:

**Option 1 — SSH (recommended)**

1. Create SSH key:
2. ssh-keygen -t ed25519 -C "your\_email@example.com"
3. Add the public key (~/.ssh/id\_ed25519.pub) to GitHub → Settings → SSH and GPG keys.
4. Set your repo remote to SSH:
5. git remote set-url origin git@github.com:khushi-2023/voice\_git\_automation\_final.git

**Option 2 — HTTPS + Credential Manager / Personal Access Token (Windows)**

1. Create a GitHub Personal Access Token (no password prompts will work long-term).
2. Use Git Credential Manager or store credentials:
3. git config --global credential.helper wincred
4. # or
5. git config --global credential.helper store
6. The first time you run git push, enter username and the PAT as password. It will be saved.

If automatic push fails due to authentication, the program will log the failure. Use SSH for the smoothest voice-driven push experience.

**D. Run the program**

1. Start the voice controller:
2. python main.py
   * It will say “Please say 'offline' or 'online' to choose ASR mode” — say offline or online.
   * In **offline** mode it uses Vosk streaming (fast) and falls back to Whisper only if Vosk returns empty.
   * In **online** mode it uses Google Web Speech only (no fallback).
3. Open the dashboard (in a separate terminal while venv active):
4. streamlit run dashboard.py
   * Visit http://localhost:8501 to see logs updating in near real-time.

**E. Common voice commands**

* commit or commit all → stages all and commits with message voice commit.
* commit main.py → stages main.py and commits with message voice commit main.py.
* commit main.py with message 'fix bug' → stages and commits with custom message.
* push → pushes (code uses git push and push\_current\_branch() when appropriate).
* pull → git pull.
* status → git status
* create branch dev → git branch dev + checkout.
* switch to dev or checkout dev → git checkout dev.
* undo last commit → git reset --soft HEAD~1
* exit / quit / stop → stop the voice loop.

**6) Extra notes & troubleshooting**

**If Vosk isn't listening or returns empty:**

* Ensure sample rate is 16000 (microphone supports it). If your microphone uses 44100, you can use sounddevice to resample or try setting samplerate=44100, but Vosk expects 16000 — better to use microphone that supports 16k or use PyAudio device selection.
* If you get repeated Listening (offline)... with no recognition, run this small check to list input devices:
* import sounddevice as sd
* print(sd.query\_devices())
* print("Default input:", sd.default.device)

Use sd.default.device = <device\_index> to force the microphone index if needed.

**If automatic push fails (auth):**

* Check git remote -v
* Try git push origin <branch> manually, fix credentials, then voice push will succeed.
* SSH method avoids repeated auth issues.

**If Whisper is slow:**

* Keep WHISPER\_MODEL="tiny" for fallback speed. Larger models improve accuracy but are much slower on CPU.

**Avoid committing logs to Git**

* .gitignore already excludes logs/ and command\_log.txt. Make sure you git rm --cached those if previously committed:
* git rm --cached command\_log.txt
* git rm -r --cached logs
* git commit -m "Remove logs from repo and add to gitignore"
* git push

**7) Final checklist before running**

1. python -m venv venv && venv\Scripts\activate (Windows) or source venv/bin/activate
2. pip install -r requirements.txt
3. python -m spacy download en\_core\_web\_sm
4. Place Vosk model in models/vosk-model-small-en-us-0.15
5. Configure GitHub auth (SSH recommended)
6. python main.py then in separate terminal streamlit run dashboard.py

If you want, I can:

* Paste a ready-to-download ZIP of the project (main.py, dashboard.py, requirements.txt, .gitignore, README).
* Create a shorter copy of main.py for low-powered machines (skip Whisper entirely).
* Add multi-file commit voice syntax like “commit main.py and dashboard.py with message 'x'”.

Tell me which of those (ZIP, lighter version, multi-file commit) you want next and I’ll produce it.