Guía de desarrollador · multifunctional-agent

multifunctional-agent: guía técnica

Cómo está construido el agente multifuncional: reglas duras, scripts de facturas y de audio, el banner aislado, el motor local de transcripción, la autoprueba y cómo agregar una funcionalidad.

3 nov 2026 | Guía de desarrollador | Claude Code | 21 min de lectura

Descripción general

El **Agente Multifuncional** (`multifunctional-agent`) resuelve tareas operativas recurrentes de Full Service que no pertenecen a una etapa del flujo comercial. Hoy tiene **dos funcionalidades** y está pensado para sumar más. Se invoca con el comando `/multifunctional-agent` y corre en **Claude Code**, gobernado por su `CLAUDE.md`.

#	Funcionalidad	Recibe	Entrega	Script principal
1	**Unificación de facturas**	Facturas o recibos (PDF o imagen)	Word y PDF en orden cronológico, una factura por hoja, bajo un banner	`scripts/facturas.py`
2	**Transcriptor de audio**	Audio o video de una reunión	Word y PDF con la transcripción por partes y hablantes	`scripts/transcribir_audio.py` y `scripts/transcripcion.py`

2 funcionalidades, cada una con su carpeta de trabajos

11 reglas duras en CLAUDE.md

1 archivo que se toca para rediseñar el banner

0 servicios externos: todo el procesamiento es local

LO ESENCIAL

Dos ideas gobiernan el diseño. **El recibo manda sobre el nombre del archivo**: fecha, hora, proveedor y tipo salen del propio recibo, mirado a la vista. Y **todo se procesa en local**: ni una factura ni el audio de una reunión salen del equipo. Lo demás (banner, motor de audio) está aislado para poder cambiarse sin tocar el resto.

Qué hace y qué no hace

El agente hace	El agente no hace
Pregunta qué funcionalidad se quiere y, con la respuesta, pide solo lo necesario	Entrevistar al usuario ni cambiar de funcionalidad por su cuenta
Ordena las facturas por lo que dice cada recibo	Fiarse del nombre del archivo ni inventar una fecha que el recibo no trae
Entrega siempre Word y PDF, generado el PDF del mismo `.docx`	Entregar un solo formato ni dos documentos que puedan diferir
Transcribe en local, separa hablantes y propone nombres	Subir un audio a un servicio externo, ni asignar un nombre que el audio no revela
Mira el PDF a la vista antes de entregar	Dar por bueno un resultado que no ha renderizado y revisado
Pide permiso antes de instalar paquetes o bajar modelos	Instalar o descargar por iniciativa propia

Cómo corre: el comando y `CLAUDE.md`

El agente vive en `agents/claude-agents/multifunctional-agent/`. Tiene dos puntos de entrada:

- `CLAUDE.md` en su carpeta: su rol, la pregunta clave, las reglas duras, el flujo de cada funcionalidad, la sección de comandos «!», el checklist de cierre y la arquitectura.
- `.claude/commands/multifunctional-agent.md`: el comando `/multifunctional-agent [facturas | audio] [carpeta o archivos]`. Indica leer `CLAUDE.md`, y decide la funcionalidad: «facturas» o recibos adjuntos → la 1; «audio», «transcribir» o un `.mp3/.mp4/.m4a/.wav` → la 2; si no, hace la pregunta clave.

Si la sesión está abierta en la raíz del repositorio, el comando le dice al agente que entre a su carpeta para trabajar y para correr los scripts.

La pregunta clave

Al empezar, o al invocarse sin decir qué se quiere, el **primer mensaje** es esta pregunta y nada más (sin pedir archivos todavía):

> ¿Con qué funcionalidad quieres proceder? 1. Unificación de facturas · 2. Transcriptor de audio.

Usa `AskUserQuestion` con las dos opciones y, si no está disponible, la lista numerada. **Excepción:** si el mensaje inicial ya deja clara la funcionalidad, no pregunta: lo dice en una línea («Procedo con la funcionalidad 1») y empieza. Al terminar ofrece la otra en una línea.

Estructura del repositorio

```
multifunctional-agent/
  CLAUDE.md                        # cómo trabaja el agente
  .claude/commands/multifunctional-agent.md  # el comando /multifunctional-agent
  recursos/                        # solo lectura
    plantilla/Plantilla Word - Campuslands.docx  # membrete (facturas y transcripción)
    fuentes/                       # Poppins (OFL), tipografía oficial
    modelos-audio/                 # modelos de diarización (fuera de git)
  scripts/
    comun.py                       # LibreOffice, fechas en español, campos de Word
    factura_banner.py              # EL BANNER: único archivo para rediseñarlo
    facturas.py                    # inspeccionar · construir
    transcribir_audio.py           # audio a segmentos.json + transcripcion.md
    transcripcion.py               # .json / .md a Word + PDF
    requisitos-audio.txt           # paquetes del motor de audio
    autoprueba.py                  # prueba de punta a punta con datos sintéticos
  trabajos/
    facturas/<AAAA-MM-DD>-<slug>/  # entrada/ · manifiesto.json · salida/
    transcripciones/<slug>/        # entrada/ · trabajo/ · salida/
  wiki/
    index.md · log.md              # catálogo de trabajos y bitácora
    banner-facturas.md             # especificación del banner y cómo cambiarlo
    formato-transcripcion.md       # estructura de la transcripción
    motor-transcripcion.md         # motor local, instalación, rendimiento y límites
```

- **Un trabajo, una carpeta.** Un lote de facturas o una reunión por carpeta, con `entrada/` (copia de lo que dio el usuario) y `salida/` (Word y PDF). El slug va en minúsculas, sin tildes ni espacios, con guiones.

- `recursos/` **es de solo lectura.** La plantilla Word, las fuentes y los modelos no se editan en un trabajo.
- **Fuera de git:** los modelos de voz (`recursos/modelos-audio/`) y los audios de entrada (`trabajos/transcripciones/*/entrada/*`), por pesados y confidenciales. Lo define el `.gitignore` del agente.

> DÓNDE VIVEN LOS RECURSOS COMPARTIDOS
>
> El logo y la marca de Campuslands no se copian aquí: están en `resources/` en la raíz del repositorio. El agente solo conserva la plantilla Word con el membrete, la tipografía Poppins y los modelos de audio.

Las reglas duras

`CLAUDE.md` fija once reglas. Las que más condicionan el código:

#	Regla	Qué significa en la práctica
1	Veracidad	Fecha, hora, proveedor y tipo salen del recibo, a la vista. En la transcripción no se inventan palabras: lo dudoso va como `[inaudible]`
2	Originales intactos	Lo del usuario se copia a `entrada/` y ahí queda; el resultado sale a `salida/`
3	Manda el recibo	Si el nombre del archivo contradice al recibo, se usa el recibo y queda una nota visible (sin horas)
4	Una factura por hoja	Solo se quita el blanco y se enmarca; todas las hojas son verticales sobre la plantilla
5	Banner vigente	Lo define `factura_banner.py` (v2). No se rediseña por iniciativa propia; el cambio se hace solo en ese archivo
6	Confidencialidad	Todo en local; nunca se sube un audio ni una factura a un servicio externo sin petición expresa
7	Instalar = pedir permiso	Se dice qué se instala y cuánto pesa, se espera el sí

#	Regla	Qué significa en la práctica
8	Transcripción literal	No se resume ni se «mejora» el habla; no está verificada de oído y se dice en la entrega
9	Hablantes y género = propuestas	Un nombre solo si el audio lo revela; Hombre/Mujer es una estimación por tono de voz; se listan como `PROPUESTO`
10	Verificación a la vista	Se renderiza el PDF y se revisa antes de entregar
11	Word y PDF siempre	El PDF sale de convertir el mismo `.docx` con LibreOffice

Funcionalidad 1: unificación de facturas



Comandos

Desde la carpeta del agente:

TERMINAL

```
python -X utf8 scripts/facturas.py inspeccionar trabajos/facturas/<lote>
python -X utf8 scripts/facturas.py construir trabajos/facturas/<lote> [--nombre X] [--sin-pdf]
```

- `inspeccionar` lee el texto de cada recibo (PDF con texto) con PyMuPDF y escribe `manifiesto.json` con una propuesta de fecha, hora, proveedor y tipo, más los avisos de detección (por ejemplo, que el nombre del archivo y el recibo no coinciden). Una imagen o un escaneo no tiene texto: la propuesta sale vacía y la única fuente es la lectura a la vista.
- `construir` normaliza el manifiesto, **ordena** por (fecha, hora, nombre del archivo), arma el `.docx`, lo convierte a PDF y comprueba que el PDF tiene tantas páginas como hojas esperadas.

El manifiesto

Es el contrato entre la detección, el agente y el constructor. Una entrada por factura:

```
manifiesto.json
  titulo: "Facturas"
  facturas[]:
    archivo: "28102026 - 13h31m - inDrive.pdf"
    fecha: "2026-10-28"           # AAAA-MM-DD
    hora: "15:31"                 # HH:MM en 24 h; null si el recibo no trae hora
    proveedor: "inDrive"          # tal como aparece en el recibo
    tipo: "Transporte"            # ver la lista de tipos
    nota: ""                      # nunca menciona horas
    paginas: 1
    _deteccion: {...}             # lo que propuso el script, incluidos los avisos
```

Reglas del manifiesto que el agente aplica al confirmar:

- **Hora:** solo sirve para ordenar y **no se imprime**. Si el recibo trae varias, se usa la del inicio del servicio (viaje, consumo). Sin hora: `null`; la factura queda al final de su día y se avisa.
- **Tipo:** `Transporte`, `Alimentación`, `Hospedaje`, `Peajes y parqueadero`, `Combustible`, `Software y nube` u `Otro`. Solo si es evidente; si no, `Otro`.
- **Nota:** se escribe cuando hay algo que decir de esa factura; nunca menciona horas.

El documento

- **Plantilla Word de Campuslands** (`recursos/plantilla/`), hoja Carta vertical, con el membrete y los márgenes de la plantilla.
- Tipografía **Poppins**, incrustada en el Word por `comun.incrustar_fuente`, y cargada desde `recursos/fuentes/` al generar el PDF, de modo que se ve igual en cualquier equipo.
- **Una factura por hoja.** El recibo se renderiza a imagen, se recorta de su blanco, se enmarca y se escala al ancho del banner ajustándolo al espacio libre. Un recibo de 2 páginas va una sobre otra si conserva al menos el 85 % de legibilidad; si no, una hoja por página con **FACTURA (CONT.)**. Desde 3 páginas, una hoja por página.
- Leyenda en cursiva de 7 pt: `Archivo original: <nombre> · <nota>`. El pie es `Pág. N/M` a la derecha, a propósito corto para no pisar la línea del membrete.

- **Ajuste automático:** `construir` reconstruye hasta 6 veces reduciendo las imágenes un 7 % cada vez si el PDF no tiene el número de páginas esperado. Si no lo logra, falla con un mensaje.
- El nombre de salida sale del rango de fechas de los recibos (`Facturas_28-29_Octubre_2026_Campuslands`).

El banner de las facturas

El banner es **un solo archivo**, `scripts/factura_banner.py`, porque el usuario lo seguirá mejorando. `facturas.py` solo llama a:

TERMINAL

```
agregar_banner(doc, indice, total, datos, ancho_twips, continuacion)
```

y no conoce colores, fuentes ni medidas.

Diseño vigente (v2)

Una **franja simétrica**: tabla de Word de cuatro celdas iguales (25 % cada una), contenido centrado, filete azul `0D497B` de 1,5 pt arriba y abajo, divisores finos `C9D6E3`, fondo `EEF3F8` (la paleta del membrete) y tipografía Poppins.

Celda	Etiqueta	Valor
1	**FACTURA** (o **FACTURA (CONT.)** en las hojas extra)	`01` en azul 13 pt y `/ N`
2	**FECHA**	«Mié 28 oct 2026»
3	**TIPO**	«Transporte»
4	**PROVEEDOR**	«inDrive»

La v1 tenía dos barras (marca y datos, con hora) y estaba desequilibrada; el usuario pidió que fuera simétrica, sin la barra superior (el logo ya lo trae el membrete) y sin hora.

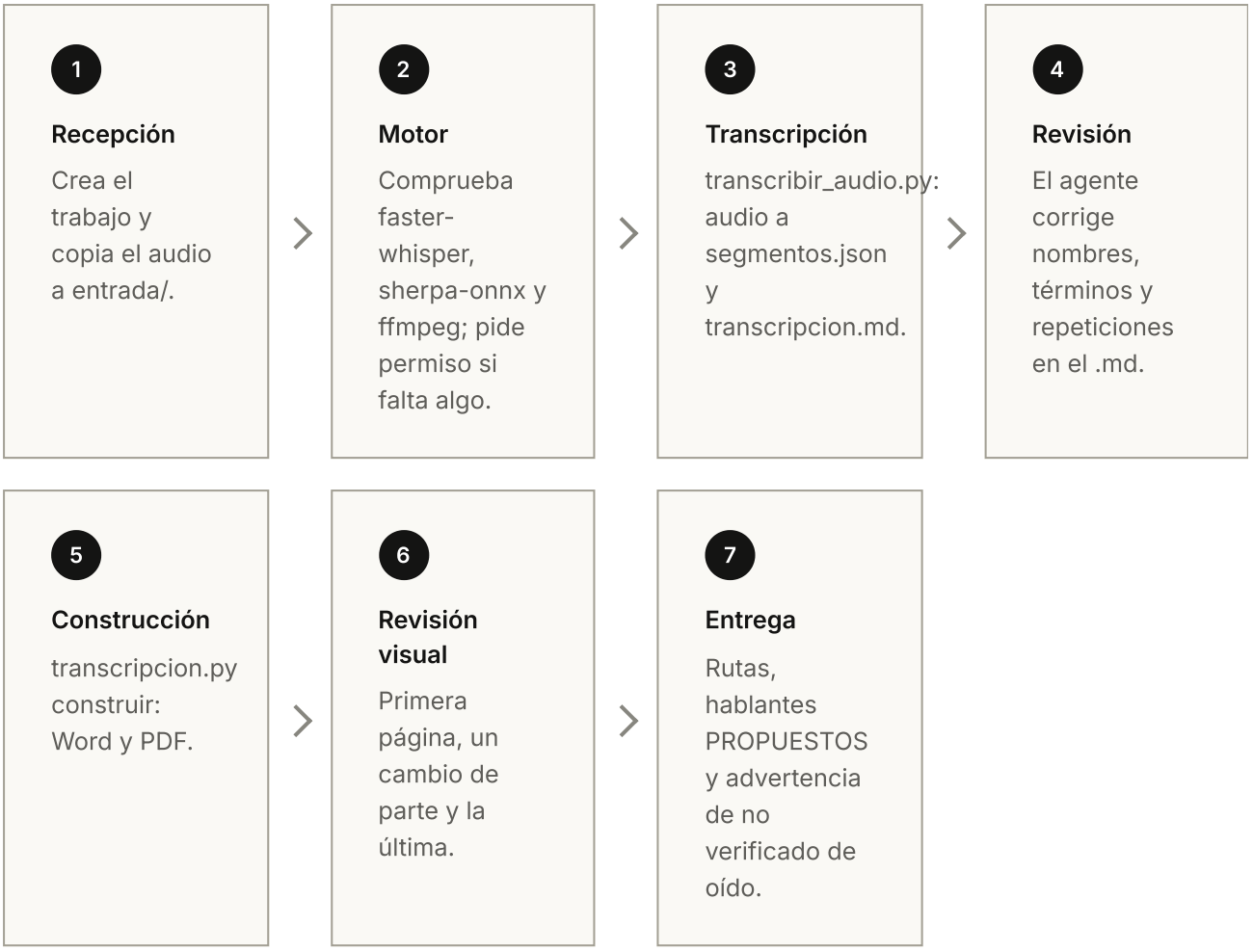
Cómo rediseñarlo

Quiero...	Qué hago
Cambiar colores, tipografía o número de columnas	Constantes al inicio del archivo: `AZUL`, `FONDO`, `DIVISOR`, `ETIQUETA`, `VALOR`, `FUENTE`, `N_COLUMNAS`
Rediseñarlo por completo (logo como imagen, otros campos, otra estructura)	Reescribir `agregar_banner` **conservando su firma**
Un banner más alto	Subir `RESERVA_VERTICAL_TW` en `facturas.py` (espacio que se reserva por hoja para banner y leyenda); `construir` reduce las imágenes solo si no caben
Mostrar un dato nuevo (por ejemplo, el total)	Agregarlo a `DatosFactura` y al manifiesto

Después, siempre: `python -X utf8 scripts/autoprueba.py` y revisar a la vista el PDF. Las comprobaciones de la autoprueba sobre el texto del banner (por ejemplo, que no aparezca la hora) tendrán que actualizarse si el diseño cambia.

> **EL AGENTE NO REDISEÑA EL BANNER POR INICIATIVA PROPIA**
>
> Es la regla dura 5. Cuando el usuario lo pide, el cambio se hace solo en `factura_banner.py` y se documenta en `wiki/banner-facturas.md`.

Funcionalidad 2: transcriptor de audio



Comandos

TERMINAL

```
python -X utf8 scripts/transcribir_audio.py <audio> --salida
trabajos/transcripciones/<slug>/trabajo --vocabulario "nombres, términos"
python -X utf8 scripts/transcripcion.py construir
trabajos/transcripciones/<slug>/trabajo/transcripcion.md --nombre "Transcripcion - <slug>"
```

Opciones de `transcribir_audio.py`:

Opción	Para qué
`--modelo`	`large-v3-turbo` (defecto), `medium`, `small`, `base`, `tiny`
`--idioma`	Idioma del audio (defecto `es`)

Opción	Para qué
`--hablantes N`	Número de hablantes, si se sabe: mejora la separación
`--umbral`	Umbral de agrupación de voces sin `--hablantes` (más alto, menos hablantes)
`--sin-diarizacion`	No separa hablantes: todo sale como «Hablante 1»
`--sin-vad`	No descarta los silencios antes de transcribir (en la prueba no mejoró y arriesga frases inventadas)
`--vocabulario`	Nombres propios y términos separados por comas, para que se escriban bien
`--parte-min`	Minutos por parte del documento (defecto 5; 0 = sin partes)
`--desde` / `--hasta`	Un tramo en segundos, para probar sin procesar todo
`--descargar-modelos`	Baja los modelos de diarización y termina si no hay audio

Opciones de `transcripcion.py construir`: `--salida`, `--nombre`, `--titulo`, `--parte-min` (solo `.json`), `--sin-ficha` (quita el cuadro de datos) y `--sin-pdf`.

La cadena de procesamiento

```
audio o video
  ffmpeg: WAV 16 kHz mono
    faster-whisper: palabras con tiempos
    sherpa-onnx: tramos «quién habla» (diarización)
  unión palabra con hablante: intervenciones
  tono de voz (F0): género estimado de cada hablante
  segmentos.json + transcripcion.md
    transcripcion.py: Word + PDF Campuslands
```

Pieza	Qué es	Peso
ffmpeg	Extrae el audio de mp3, mp4, m4a, wav, ogg, webm, mkv…	Programa del sistema (`winget install Gyan.FFmpeg`)
faster-whisper	Whisper optimizado para CPU. Modelos `small` (≈ 0,5 GB), `medium` (≈ 1,5 GB) y `large-v3-turbo` (≈ 1,6 GB, el defecto)	`pip`; el modelo se baja la primera vez que se usa
sherpa-onnx	Diarización: segmentación `pyannote 3.0` y huella de voz `3D-Speaker ERes2Net`	≈ 45 MB, con `--descargar-modelos`, en `recursos/modelos-audio/`

A Whisper se le pasa el arreglo de muestras y no la ruta del archivo: su decodificador interno (PyAV) cambia entre versiones y rompió con `av 19`.

Qué decide el script

- **Intervenciones:** se agrupan palabras del mismo hablante; una pausa de más de 1,2 s abre una nueva; ninguna pasa de unos 28 s (se corta en el siguiente fin de frase); nunca cruza el corte de una parte.
- **Hablantes pequeños:** el que tiene menos de max(8 s, 1,5 % de la duración) de voz suele ser ruido o un corte de otro: sus palabras pasan al hablante vecino.
- **Género:** mediana de F0 de cada hablante. Menos de 155 Hz → Hombre; más de 175 Hz → Mujer; entre ambos, sin género. **Es una estimación**, no un dato.
- **Nombres:** el script **nunca** los pone; los asigna el agente en la revisión, solo si el audio los revela.

Formatos de entrada y salida

`transcripcion.py` acepta dos entradas:

- `.md`, el formato de la referencia. Se puede escribir o corregir a mano. Un comentario opcional `<!-- meta: {...} -->` alimenta el cuadro de datos (audio, duración, fecha, tamaño de parte).
- `.json`, lo que escribe `transcribir_audio.py`: título, audio, duración, fecha, `parte_s`, un mapa de hablantes (`nombre`, `genero`) y los segmentos con tiempos **absolutos** en segundos.

Con el `.json` se puede cambiar `--parte-min` y re-partir el documento sin volver a transcribir. Con el `.md`, cualquier corrección de texto se aplica volviendo a correr `construir`.

La estructura del documento es **la de la referencia, sin cambios**: título, «Parte N» (una por 5 minutos, tiempos que reinician en `00:00`) y una línea por intervención `[mm:ss - mm:ss] Hablante N (X): texto`, donde `X` es el nombre, `Hombre` o `Mujer`. Lo único que se agrega es el cuadro de datos bajo el título.

Elemento	Referencia	Campuslands
Página	A4 sin membrete	Plantilla Word de Campuslands, con membrete
Título	Helvetica 20 negrita	Calibri 22 negrita azul `0D497B`, filete verde `5DAC6D`
«Parte N»	Helvetica 14 negrita azul	Calibri 14 negrita azul, estilo Título 1 de Word (aparece en el panel de navegación)
Intervención	Tiempo y hablante en negrita negra	Tiempo en verde `2F7D45`, hablante en azul, texto en `262626`
Pie	Título y «Página N»	«Página N» a la derecha

Instalar el motor de audio

Las facturas y el armado de documentos solo necesitan Python, LibreOffice y unas librerías. **Solo el audio** necesita el motor local, que el agente instala **únicamente con permiso** (regla 7).

Requisito	Para qué	Cómo
Python con `python-docx`, `PyMuPDF`, `Pillow`, `lxml`	Facturas y construcción de documentos	`pip install`
LibreOffice	Convertir Word a PDF	`winget install TheDocumentFoundation.LibreOffice`
ffmpeg	Leer el audio o el video	`winget install Gyan.FFmpeg`
`faster-whisper`, `sherpa-onnx`, `numpy`	Transcripción y diarización	`pip install -r scripts/requisitos-audio.txt`

Requisito	Para qué	Cómo
Modelos de diarización (≈ 45 MB)	Separar hablantes	`python -X utf8 scripts/transcribir_audio.py --descargar-modelos`

El agente comprueba el motor con:

```
TERMINAL
python -c "import faster_whisper, sherpa_onnx"
ffmpeg -version
```

Todos los scripts se corren **desde la carpeta del agente** y con `python -X utf8` (en Windows la consola no es UTF-8 por defecto).

Rendimiento medido

Prueba del 2026-11-06 con los primeros 5 minutos de una reunión real de unos 51 minutos (sala con varios participantes), solo CPU y con el vocabulario del proyecto. Se contrastó con una transcripción de referencia hecha con otra herramienta. La «diferencia de palabras» no es una medida de error absoluta (la referencia tampoco es la verdad, trae muletillas que Whisper omite): sirve para **comparar modelos entre sí**.

Modelo	Tiempo (5 min, con diarización)	Diferencia de palabras	Observaciones
`small`	≈ 190 s	≈ 49 %	Oye mal nombres y términos técnicos, inventa palabras y confunde hablantes. Sirve de borrador, no de entrega
`large-v3-turbo` **(defecto)**	≈ 310 s (≈ 1× la duración del audio)	≈ 44 %	Frases legibles y con sentido; separa bien las voces. Aún necesita revisión en nombres propios y cifras
`large-v3-turbo` con `--sin-vad`	≈ 270 s	≈ 45 %	Sin mejora medible: se deja el filtro de silencios activado

El motor da un **primer borrador legible**, no una transcripción final. Por eso el agente la revisa (regla 8) y la entrega diciendo que no está verificada de oído.

Límites que siempre se dicen

1. El agente **no puede escuchar el audio**: no puede verificar la transcripción de oído. Se entrega como automática y se sugieren 2 o 3 marcas de tiempo para contrastar.
2. Con audio de sala (varias personas a distancia, voces que se pisan) baja la precisión y la diarización mezcla hablantes. Mejora con micrófono cercano, un hablante a la vez y `--hablantes N`.
3. No detecta risas ni gestos: el motor no genera `[Risas]` ni `[Asiente]`, y el agente no los inventa.
4. El género es una estimación por tono de voz; los nombres son propuestas hasta que el usuario las confirme.

Si el resultado sale flojo

1. Repetir con otro modelo: `--modelo medium` (`small` es más rápido pero peor).
2. Pasar `--vocabulario` con los nombres propios y términos del proyecto.
3. Pasar `--hablantes N` si se sabe cuántos participan.
4. Si nada alcanza: transcribir con otra herramienta y entregarle al agente el texto en el formato de la referencia (`.md`); él arma el documento.

La autoprueba

TERMINAL

```
python -X utf8 scripts/autoprueba.py [--conservar]
```

Prueba de punta a punta con **datos sintéticos**: no usa archivos reales ni el motor de audio. Sale con código 0 si todo pasa.

Parte	Qué genera y qué comprueba
Facturas	Recibos falsos (PDF con texto, uno de 3 páginas, una imagen sin texto, uno sin hora y otro con nombre de archivo engañoso). Corre `inspeccionar` y `construir` y comprueba la hora y la fecha detectadas, que una imagen sin texto no inventa fecha, el orden cronológico, el número de páginas, la nota de discrepancia, que la hora no se imprima, el pie «Pág. N/M» y la tipografía Poppins en el PDF

Parte	Qué genera y qué comprueba
Transcripción	Un `.json` de unos 12 minutos con dos hablantes. Lo convierte a Word y PDF y comprueba las 3 partes, que los tiempos reinicien, las etiquetas de hablante, que ninguna línea cruce el corte y la ida y vuelta `.json` a `.md` a Word

Se corre **siempre que se toque código** (es parte del checklist de cierre). `--conservar` deja los archivos generados para inspeccionarlos.

Control de calidad

Antes de dar un trabajo por cerrado, el agente repasa este checklist:

1. Word **y** PDF en `salida/`, y los miró a la vista.
2. Nada de lo que dio el usuario fue modificado; `entrada/` quedó como llegó.
3. Los avisos (discrepancias de nombre, facturas sin hora, hablantes sin nombre, transcripción no verificada de oído) están en el mensaje de entrega.
4. Una entrada en `wiki/log.md` y una fila en `wiki/index.md`.
5. Si tocó código, `autoprueba.py` pasa.

La revisión visual es parte de la entrega, no un extra. En facturas, el PDF se renderiza con PyMuPDF (`page.get_pixmap`) y se mira cada página: banner correcto, recibo completo y legible, pie «Página N de M». En la transcripción, la primera página, un cambio de parte y la última.

Comandos «!»

Los comandos son órdenes cortas del usuario (`!optimize`, `!notime`…) que el agente debe ejecutar de forma obligatoria, siempre dentro de sus reglas duras. Hay un solo catálogo para todos los agentes y de él salen las instrucciones de cada uno, las bitácoras y las tablas de las guías.

Dónde vive cada cosa

Qué	Dónde
Catálogo, regla común y qué comandos tiene cada agente	`resources/comandos/comandos.py` (la fuente única; no se edita a mano nada de lo que genera)

Qué	Dónde
Catálogo legible y matriz por agente	`resources/comandos/catalogo.md`
Instrucciones del agente	sección «Comandos (`!`)» de su `CLAUDE.md` , entre las marcas `<!-- comandos:inicio -->` y `<!-- comandos:fin -->`
Tabla de esta guía	se genera del mismo catálogo al compilar las guías

Cómo añadir o cambiar un comando

1. Edita `CMDS` (definición), `AGENTES` (a quién aplica y su nota) o `ALIAS` en `resources/comandos/comandos.py` .
2. Corre `python -X utf8 resources/comandos/comandos.py` : reescribe la sección de cada agente, el catálogo y, la primera vez, añade la entrada a su bitácora.
3. Recompila las guías (`python guides/_src/build.py`).
4. `python resources/comandos/comandos.py --verificar` falla si algo quedó desfasado.

En este agente, los comandos que dependen de su trabajo tienen una nota propia: `!analyze` revisa los archivos de entrada sin generar nada, `!audit` busca discrepancias entre nombre de archivo y recibo, `!visual-audit` revisa el PDF página por página, `!security-check` comprueba la confidencialidad, y `!agent-handoff` prepara el traspaso de una transcripción de reunión con cliente a `functional-scope` como fuente de `alcances/<cliente>/fuentes/` . No aplican `!requirements` ni `!breakdown` .

Comandos de este agente

General

Comando	Qué hace aquí
`!analyze`	Analiza la solicitud, el problema, los documentos o la situación **antes de actuar**: qué se pide, qué hay, qué falta y qué riesgos ve. Entrega hallazgos; no ejecuta ni modifica nada. **Aquí:** Revisa los archivos de entrada y dice qué detectó (facturas: fechas, horas y proveedores; audio: duración y motor disponible) sin generar nada todavía.
`!research`	Investiga a fondo el tema, cruza fuentes y entrega lo relevante con **cada dato citado a su fuente**. Sin fuente verificable no es un dato: se marca `[PENDIENTE]` .

Comando	Qué hace aquí
`!plan`	Entrega un plan de trabajo estructurado (pasos, orden, insumos, entregables, supuestos y puntos de decisión) y **se detiene a esperar aprobación** antes de ejecutarlo. **Aquí:** Presenta cómo procesará el lote o el audio (orden esperado, modelo de voz, dudas) y espera aprobación antes de generar.
`!execute`	Ejecuta ya lo solicitado o el siguiente paso aprobado del flujo, sin volver a proponer ni a preguntar lo ya decidido. No se salta las puertas de aprobación ni de verificación.
`!review`	Revisa el trabajo indicado y entrega oportunidades de mejora priorizadas (qué cambiaría y por qué). No lo modifica salvo que se le pida.
`!audit`	Auditoría integral de punta a punta: contradicciones entre documentos, cifras que no cuadran, reglas incumplidas y datos sin fuente. Entrega hallazgos con evidencia y severidad. **Aquí:** Busca discrepancias entre el nombre de cada archivo y lo que dice el recibo, fechas dudosas, facturas sin hora y hablantes sin respaldo en el audio.
`!validate`	Comprueba, requisito por requisito, que el resultado cumple lo establecido (reglas duras, insumos, formato). Veredicto «cumple / no cumple» por ítem, con evidencia. **Aquí:** Facturas: orden cronológico por el recibo, una factura por hoja y banner igual al recibo. Transcripción: estructura de la referencia, partes de 5 min y etiquetas de hablante.
`!optimize`	Optimiza lo que se esté desarrollando o ya existe (proceso, solución o resultado): mejora claridad, calidad, consistencia, peso o rendimiento **sin cambiar su alcance ni sus datos**. **Aquí:** Mejora la calidad del resultado (p. ej. repetir la transcripción con un modelo más preciso o con vocabulario) sin cambiar el contenido ni el diseño vigente.
`!compare`	Compara alternativas con criterios explícitos y recomienda la más conveniente, con el porqué.
`!summarize`	Resumen claro y estructurado (lo esencial, decisiones y pendientes) del trabajo, de un documento o de la conversación.
`!document`	Documenta lo hecho: actualiza la wiki y la bitácora del agente, o genera la documentación técnica, funcional o ejecutiva que se pida.
`!prioritize`	Clasifica actividades, hallazgos o elementos por impacto, importancia y urgencia, y entrega el orden recomendado.

Comando	Qué hace aquí
`!help`	Lista los comandos disponibles en este agente, agrupados, con su función.

Modos

Comando	Qué hace aquí
`!notime`	**Modo prisa:** apresura el desarrollo optimizándolo: agrupa preguntas en un solo mensaje, paraleliza, reutiliza lo ya validado, omite explicaciones y pasos opcionales y responde solo con lo esencial. **Nunca** se salta verificaciones obligatorias, puertas de aprobación ni la veracidad: acelera el camino, no recorta la calidad.
`!think-deep`	Análisis más profundo antes de responder: considera alternativas, casos límite y consecuencias, y muestra solo el razonamiento que importa.
`!critical-mode`	Modo crítico: cuestiona decisiones, supuestos y su propio trabajo; busca errores y contradicciones antes de entregar y los dice.
`!no-assumptions`	No supone: ante un dato faltante o ambiguo pregunta (todas las dudas juntas) en lugar de rellenar o inferir.
`!double-check`	Segunda verificación independiente de su propio resultado (releer la fuente, recalcular, comparar) y corrección de lo que encuentre; reporta qué verificó. **Aquí:** Facturas: relee cada recibo y lo compara con su banner. Transcripción: contrasta una muestra de líneas con sus marcas de tiempo.
`!strict-mode`	Sigue al pie de la letra los requisitos e instrucciones dados: sin iniciativas, adiciones ni interpretaciones creativas.

Técnico y seguridad

Comando	Qué hace aquí
`!debug`	Identifica la causa de un error de software (script, conversión, verificación) y lo corrige; vuelve a correr la verificación para comprobarlo. **Aquí:** Corrige fallos de los scripts (LibreOffice, ffmpeg, modelos de voz, lectura de un recibo) y repite la verificación.
`!troubleshoot`	Investiga un problema técnico y determina su **causa raíz**, con evidencia. Diagnostica; no cambia nada.

Comando	Qué hace aquí
`!security-check`	Revisa seguridad y confidencialidad: datos de otros clientes mezclados, credenciales, datos personales, información que no debe salir del proyecto. **Aquí:** Confidencialidad: recibos con datos personales y audios de reuniones con clientes; comprueba que nada salió del equipo (todo es local) ni quedó versionado en git.

UX/UI y diseño

Comando	Qué hace aquí
`!visual-audit`	Examina proporciones, distribución, tipografía y consistencia visual del entregable, a la vista y página por página. **Aquí:** Renderiza el PDF y lo revisa página por página: banner, recibo completo y legible, pie «Página N de M»; en la transcripción, partes, hablantes y membrete.
`!consistency-check`	Detecta inconsistencias entre secciones, componentes, cifras, nombres y estilos del mismo entregable.

Gestión de proyecto

Comando	Qué hace aquí
`!track-progress`	Informa el progreso y los pendientes (hecho, en curso, bloqueado, `[PENDIENTE]`) a partir de la bitácora y del trabajo actual.
`!generate-report`	Genera un informe ejecutivo (decisiones, resultado, pendientes) o técnico (método, evidencias, verificaciones) del trabajo.
`!final-delivery`	Prepara y valida la entrega final: corre todas las verificaciones obligatorias, completa el checklist de cierre (wiki y bitácora) y entrega solo si todo pasa. **Aquí:** Word y PDF verificados a la vista, avisos incluidos, wiki y bitácora al día y la autoprueba superada si se tocó código.
`!agent-handoff`	Prepara el traspaso al siguiente agente del flujo comercial: resumen, entregables con su ruta, decisiones tomadas y pendientes. **Aquí:** Prepara el traspaso de una transcripción de reunión con cliente a **Alcances Funcionales** (`functional-scope`) como fuente de `alcances/<cliente>/fuentes/` .

IA avanzada

Comando	Qué hace aquí
`!second-opinion`	Un subagente independiente evalúa el resultado **sin ver tu razonamiento**; se contrasta con el tuyo y se explican las diferencias.
`!cross-review`	Otro especialista (subagente con otro rol) revisa el resultado y devuelve observaciones; se aplican las válidas y se informa de las descartadas.

Multiagente

Comando	Qué hace aquí
`!orchestrate`	Reparte partes **independientes** del trabajo entre varios subagentes en paralelo y las integra en un solo resultado verificado. **Aquí:** Reparte lotes de facturas o tramos de audio independientes entre subagentes; la integración y la verificación final las hace el agente principal.
`!team-review`	Varios revisores con roles distintos evalúan por separado el mismo resultado y se consolida su evaluación. **Aquí:** Revisores con rol **datos (fechas, horas, orden)**, **diseño (banner y membrete)** y **fidelidad (transcripción literal)**.
`!consensus`	Consolida las evaluaciones disponibles (de subagentes o de revisiones previas) en una sola recomendación, con las discrepancias a la vista.

Cómo agregar una funcionalidad

El agente se pensó para crecer. Una funcionalidad nueva sigue siempre estos cinco pasos:

1. **Un script** en `scripts/` que reutilice `comun.py` y, si produce documentos, genere **Word y PDF**.
2. **Una carpeta** `trabajos/<funcionalidad>/` con la misma lógica `entrada/` a `salida/`.
3. **Una sección «Funcionalidad N»** en `CLAUDE.md` (insumos, script, pasos y entrega), una fila en la tabla de funcionalidades y **una opción más en la pregunta clave**. El comando `/multifunctional-agent` también debe conocerla.
4. **Una página en** `wiki/` con el diseño de referencia, y su entrada en `wiki/log.md`.
5. **Actualizar la documentación**: el `README.md` de la raíz, estas guías (`guides/_src/contenido/multifunctional-agent/`) y, si hace falta, `resources/comandos/comandos.py`.

REUTILIZA LO QUE YA EXISTE

`comun.py` ya resuelve la conversión con LibreOffice, las fechas en español, los campos de Word (como «Pág. N/M») y la tipografía incrustada. Un script nuevo que genere un documento de Campuslands debería partir de la misma plantilla Word.

Mantenimiento

Al terminar cada trabajo, el agente añade una entrada a `wiki/log.md` con el formato `## [AAAA-MM-DD] <facturas|transcripción> | <título>` y una fila en `wiki/index.md`.

Quiero...	Qué tocar
Rediseñar el banner de las facturas	`scripts/factura_banner.py` ; luego `autoprueba.py` , revisión visual y `wiki/banner-facturas.md`
Que se reconozca un proveedor nuevo	La lista `_PROVEEDORES` de `facturas.py` (patrón, nombre y tipo). Solo se agrega lo que se reconoce con seguridad; lo demás lo completa el agente mirando el recibo
Agregar un tipo de factura	La lista de tipos en `CLAUDE.md` (el tipo es texto libre en el manifiesto)
Cambiar el formato de la transcripción	`scripts/transcripcion.py` y `wiki/formato-transcripcion.md` , en el mismo cambio
Cambiar el modelo o los parámetros de audio	`scripts/transcribir_audio.py` y `wiki/motor-transcripcion.md` (con una nueva medición)
Incorporar una regla nueva del usuario	`CLAUDE.md` y la wiki correspondiente, en el mismo cambio
Cambiar los comandos	`resources/comandos/comandos.py` , y recompilar guías y agentes

Problemas frecuentes

Síntoma	Causa probable	Qué hacer
El orden de las facturas parece al revés	Se confió en el nombre del archivo	El orden sale de fecha y hora del recibo; corregir el manifiesto y volver a `construir`
Una factura queda al final de su día	El recibo no trae hora (`hora: null`)	Es lo esperado; el agente lo avisa. Si la hora es legible en el recibo, agregarla al manifiesto
`construir` **reduce las imágenes varias veces**	Los recibos no caben con el banner en una hoja	Es el ajuste automático; si no lo consigue en 6 intentos, revisar las imágenes de entrada o `RESERVA_VERTICAL_TW`
El PDF no se genera o falla la conversión	LibreOffice no está instalado o no se encuentra	Instalar LibreOffice (`winget install TheDocumentFoundation.LibreOffice`)
La transcripción falla al abrir el audio	Falta `ffmpeg` o el formato es raro	Comprobar `ffmpeg -version`; convertir el archivo a `.mp3` o `.wav`
`import faster_whisper` **falla**	El motor de audio no está instalado	`pip install -r scripts/requisitos-audio.txt`, con permiso del usuario
Todos los hablantes salen como «Hablante 1»	Se usó `--sin-diarizacion` o faltan los modelos	Correr `--descargar-modelos` y repetir
Más hablantes de los que hay	Audio de sala; la diarización parte una voz en dos	Repetir con `--hablantes N`
Nombres propios mal escritos	El modelo no los conoce	Repetir con `--vocabulario` y revisar el `.md`
La transcripción tarda mucho	El modelo grande en CPU dura casi lo que el audio	Es lo esperado; lanzarla en segundo plano. `--modelo small` solo para borradores

Síntoma	Causa probable	Qué hacer
Frases repetidas o inventadas en un silencio	Alucinación de Whisper	Corregir en el `.md`; no desactivar el filtro de silencios (`--sin-vad`)
Un audio quedó versionado en git	Se copió fuera de `entrada/`	Retirarlo del índice; el `.gitignore` solo cubre `trabajos/transcripciones/*/entrada/*`

Cosas que no están alineadas

- Las comprobaciones de `autoprueba.py` sobre el banner (por ejemplo, que no aparezca la hora o que el texto traiga «FACTURA 01/5» y la fecha abreviada con el día) dependen del diseño v2. Si el usuario rediseña el banner, la autoprueba puede fallar sin que haya un error real: hay que actualizarla en el mismo cambio.
- La entrada `multifunctional-agent` de `resources/comandos/comandos.py` debe estar en el repositorio para que estas guías se compilen: el bloque de comandos de cada guía se genera desde ese catálogo.

Referencia rápida de archivos

Archivo	Para qué lo abres
`CLAUDE.md`	Las reglas duras, la pregunta clave y el flujo de cada funcionalidad
`.claude/commands/multifunctional-agent.md`	El comando `/multifunctional-agent` y cómo decide la funcionalidad
`scripts/factura_banner.py`	Rediseñar el banner de las facturas
`scripts/facturas.py`	Detección, manifiesto, orden y construcción del documento de facturas
`scripts/transcribir_audio.py`	El motor: audio a texto con hablantes
`scripts/transcripcion.py`	Texto o `.json` a Word y PDF de la transcripción

Archivo	Para qué lo abres
`scripts/comun.py`	LibreOffice, fechas en español, campos de Word y fuente incrustada
`scripts/autoprueba.py`	Probar todo con datos sintéticos
`wiki/banner-facturas.md`	Especificación del banner y cómo cambiarlo
`wiki/formato-transcripcion.md`	Estructura de la transcripción y formato del `.md` y el `.json`
`wiki/motor-transcripcion.md`	Instalación, rendimiento medido y límites del motor local
`wiki/index.md` y `wiki/log.md`	Catálogo de trabajos y bitácora