Guía de desarrollador · contract-perfectionist

contract-perfectionist: guía técnica

Cómo redacta el agente el contrato final de un proyecto, qué insumos usa, cómo se monta el Word sobre la plantilla corporativa y qué controles debe pasar antes de entregarse.

3 nov 2026 | Guía de desarrollador | Claude Code | 15 min de lectura

Descripción general

Perfeccionador de Contratos (`contract-perfectionist`) redacta el **contrato final** de un proyecto a partir de tres insumos: los **alcances** (qué se va a construir), la **cotización** (valores, forma de pago y cronograma) y el **contrato madre** (la plantilla estructural con las cláusulas estándar de la empresa). El resultado es un contrato en `contratos/<cliente>/` donde ambas partes quedan enteradas del objeto, el alcance, el cronograma, los valores y las reglas del proyecto, **sin ambigüedad**.

Esta guía explica cómo está construido: los insumos, el flujo de seis pasos, las nueve reglas duras, la arquitectura de cláusulas, el script que monta el documento Word y ejecuta los controles, la verificación cruzada y cómo mantenerlo. Está pensada para quien mantiene el agente.

3 insumos obligatorios: alcances, cotización y contrato madre	6 pasos del flujo, de la ingesta a la entrega
9 reglas obligatorias	2 controles automáticos antes de entregar: placeholders y diseño

LO ESENCIAL

Nunca inventa información. Todo dato específico (partes, NIT, representante legal, alcance, valores, plazos, forma de pago) debe provenir de los archivos de `recursos/` ; si no aparece, se deja un `[PENDIENTE: …]` explícito y se reporta. Y **ningún contrato se entrega sin pasar el gate** `build_contract.py check` .

Qué hace y qué no hace

El agente hace	El agente no hace
Redacta un contrato formal, neutral y específico del proyecto	Completar datos «a criterio» ni asumir la interpretación más probable
Reutiliza la arquitectura de cláusulas del contrato madre y sus cláusulas de protección	Copiar el contrato madre a ciegas: el contenido se ajusta a alcances y cotización
Pregunta al usuario ante cualquier ambigüedad, antes de escribir esa parte	Mezclar información de un cliente en el contrato de otro, ni mencionar otros clientes
Monta el `.docx` siempre sobre la plantilla corporativa, con líneas de firma reales	Generar el contrato como un documento nuevo desde cero, sin membrete
Deja un rastro auditable de la verificación cruzada	Reportar «listo» un contrato que no pasó el gate

Cómo corre: Claude Code y `CLAUDE.md`

Corre en **Claude Code** y se gobierna con `CLAUDE.md`, que define los tres insumos, las reglas duras, el flujo, la estructura del proyecto y la convención de nombres. Los cambios al propio agente (scripts, reglas, estructura) se registran en `CHANGELOG.md`; los contratos generados o revisados, en `wiki/log.md`: **no se mezclan**.

El script `scripts/build_contract.py` solo hace el **montaje mecánico**. El contenido de cada cláusula lo sigue redactando el agente en conversación con el usuario.

Estructura del repositorio

```
contract-perfectionist/
  CLAUDE.md                       # reglas duras, flujo y convención de nombres
  CHANGELOG.md                    # cambios al agente (no a los contratos)
  recursos/                       # insumos de entrada; solo lectura
    alcances/<cliente-slug>/      # documentos de alcance del proyecto
    cotizaciones/<cliente-slug>/  # cotización aprobada
    contrato-madre/               # plantilla estructural y cláusulas estándar
    Contrato - Plantilla Base.docx  # plantilla de DISEÑO con el membrete
  scripts/
    build_contract.py             # build y check
  contratos/
```

```
    <cliente-slug>/
      Contrato - <Cliente> - <YYYY-MM-DD>.docx
      datos.json                  # datos de trabajo del contrato
      verificacion-cruzada.md     # rastro de la comparación
  wiki/
    index.md                      # catálogo de contratos, con su estado
    log.md                        # bitácora de contratos
    datos-empresa.md              # datos legales propios de Campuslands
    glosario-clausulas.md         # categorías de cláusulas y su historial
    plantilla-verificacion-cruzada.md
    clientes/<cliente-slug>.md    # ficha de cada cliente
```

DÓNDE VIVE EL AGENTE Y SUS RECURSOS COMPARTIDOS

El agente está en `agents/claude-agents/contract-perfectionist/`. Los logos, el isotipo, el favicon, el `Brief Fullservice.pdf` y el Brandbook de Campuslands ya no se copian dentro de cada agente: están en `resources/campuslands-resources/`, en la raíz del repositorio (desde la carpeta del agente: `../../../resources/campuslands-resources/`).

DOS PLANTILLAS CON ROLES DISTINTOS

El **contrato madre** (`recursos/contrato-madre/`) es la plantilla de **contenido legal**: aporta la arquitectura de cláusulas y las cláusulas de protección. `Contrato - Plantilla Base.docx` es la plantilla de **diseño corporativo**: aporta el membrete, los márgenes y la tipografía. No se confunden.

Los insumos

Insumo	Dónde vive	Qué aporta
Alcances	`recursos/alcances/<cliente>/`	Qué se va a construir: módulos, integraciones, exclusiones
Cotización	`recursos/cotizaciones/<cliente>/`	Valores, forma de pago, cronograma y fases
Contrato madre	`recursos/contrato-madre/`	Arquitectura de cláusulas y protecciones estándar de la empresa

Un contrato nunca se redacta sin al menos un archivo de alcance y uno de cotización para ese cliente, además del contrato madre vigente. `recursos/` es la única fuente de datos: el agente toma de allí, o de lo que el usuario aporte directamente en la conversación, y nunca inventa.

El flujo de trabajo



Paso	Qué ocurre	Salida
1 · Ingesta	Lee alcances y cotización del cliente, el contrato madre vigente y la wiki (`index` , `datos-empresa` y, si existe, la ficha del cliente) para no repetir preguntas ya resueltas	Contexto
2 · Checklist de datos	Extrae y lista razón social, NIT, representante legal, objeto, alcance funcional detallado, cronograma y fases, valor total, forma de pago, mantenimiento y garantías. Marca lo faltante o ambiguo y **pregunta antes de continuar**	Preguntas al usuario
3 · Borrador	Redacta siguiendo la arquitectura del contrato madre e inserta el contenido del proyecto; conserva las cláusulas de protección salvo indicación contraria. Ensambla con `build_contract.py build`	`.docx` y `datos.json`

Paso	Qué ocurre	Salida
4 · Verificación cruzada	Llena la plantilla de verificación comparando el borrador con las fuentes: cero discrepancias y con rastro	`verificacion-cruzada.md`
5 · Gate de calidad	Corre `build_contract.py check`. Si falla, corrige y repite	Gate aprobado
6 · Entrega	Guarda el contrato final, actualiza `wiki/index.md` con el estado, añade la línea en `wiki/log.md` y crea o actualiza la ficha del cliente	Contrato final

Las nueve reglas duras

#	Regla	Qué significa en la práctica
1	Nunca inventar información	Todo dato específico viene literalmente de `recursos/`; si falta, `[PENDIENTE: NIT del cliente]` y se reporta
2	Consultar ante cualquier ambigüedad	Si un alcance, una cotización o una cláusula admite más de una lectura, se detiene y pregunta antes de escribir esa parte
3	El contrato madre es inspiración estructural	Se reutiliza su arquitectura y sus protecciones, pero el contenido de cada cláusula se ajusta al proyecto
4	Profesionalismo y discreción	Redacción formal y neutral, sin lenguaje comercial ni promesas no respaldadas; cada proyecto es reservado
5	Claridad para ambas partes	Debe quedar claro qué se entrega, cuándo, por cuánto, bajo qué condiciones y qué pasa si algo falla; precisión sobre elegancia
6	Diseño corporativo obligatorio	Todo `.docx` se construye **sobre** `Contrato - Plantilla Base.docx`; hacerlo desde cero es un defecto de entrega
7	Líneas de firma físicas	Línea horizontal sobre el nombre de cada representante legal, con espacio para firmar a mano o insertar firma digital

#	Regla	Qué significa en la práctica
8	Gate de entrega obligatorio	Ningún `.docx` pasa a `contratos/<cliente>/` sin superar `build_contract.py check`
9	Datos de clientes reservados, también en el repositorio	`recursos/alcances/` y `recursos/cotizaciones/` traen NIT, representantes y cifras: antes de compartir o subir el repo a un remoto, se confirma con el usuario si deben excluirse

La arquitectura de cláusulas

El contrato madre aporta la arquitectura: consideraciones, objeto, naturaleza jurídica, alcance, cronograma, valor, forma de pago, mantenimiento y soporte, garantías, obligaciones de cada parte, aceptación de entregables, datos personales, confidencialidad, terceros, suspensión y terminación, fuerza mayor, cláusula penal, vigencia, ley aplicable y firmas. `wiki/glosario-clausulas.md` la cataloga como checklist, indicando **de dónde sale el contenido** de cada categoría:

Categoría	Qué cubre	Origen del contenido
Partes y consideraciones	Identificación legal de ambas partes y motivación	Datos del cliente (a confirmar) y resumen de la cotización
Objeto	Qué se compromete a entregar el contratista	Alcances
Naturaleza jurídica	Tipo de obligación (de medio o de resultado)	Contrato madre
Alcance funcional	Módulos, funcionalidades e integraciones incluidos y excluidos	Alcances
Cronograma y fases	Plazos, hitos y fases	Cotización y alcances
Valor y forma de pago	Monto, hitos de pago y moneda	Cotización
Mantenimiento y soporte	Condiciones posteriores a la entrega	Cotización y contrato madre

Categoría	Qué cubre	Origen del contenido
Infraestructura y consumo	Quién asume costos de infraestructura y servicios	Cotización y alcances
Garantía	Duración y alcance posterior a la entrega	Contrato madre y cotización
Obligaciones de cada parte	Qué debe hacer el contratista y qué el cliente	Contrato madre y particularidades del proyecto
Aceptación de entregables	Cómo y en qué plazo se valida cada entrega	Contrato madre
Propiedad intelectual y licencia de uso	Titularidad del software y licencia otorgada	Contrato madre
Datos personales	Roles de responsable y encargado, y compromisos de tratamiento	Contrato madre y normativa aplicable
Reserva de información	Alcance y vigencia de la obligación de reserva	Contrato madre
Servicios de terceros	Deslinde de responsabilidad por proveedores externos	Contrato madre y terceros de los alcances
Suspensión y terminación	Causales y efectos	Contrato madre
Fuerza mayor	Deslinde por eventos externos	Contrato madre
Cláusula penal	Consecuencia económica de un incumplimiento grave	Contrato madre (validar porcentaje o monto si cambia)
Vigencia	Duración total del acuerdo	Cotización y condiciones de mantenimiento
Ley aplicable y controversias	Jurisdicción y mecanismo de resolución	Contrato madre

Categoría	Qué cubre	Origen del contenido
Firmas	Cierre y validez de firmas electrónicas	Contrato madre

Las cláusulas de protección estándar (propiedad intelectual, reserva de información, terceros, fuerza mayor, terminación, cláusula penal y ley aplicable) se mantienen **salvo que el usuario indique lo contrario** para ese proyecto.

Ajustes validados por proyecto

Cuando un proyecto requiere una variación reutilizable de una cláusula estándar y el usuario la valida, se documenta en la sección «Ajustes validados por proyecto» del glosario, con fecha y motivo. Por ejemplo, el estándar otorga solo licencia de uso del software; para un cliente concreto, el usuario validó que la propiedad del código y los derechos patrimoniales se **cedieran**, y quedó registrado.

Historial del contrato madre

Cada vez que se reemplaza el contrato madre se añade una entrada en «Historial de versiones» del glosario, con fecha, qué cambió y por qué, para saber si un contrato ya generado quedó desactualizado frente a la versión vigente.

El script de montaje

`scripts/build_contract.py` tiene dos subcomandos y requiere solo `python-docx` (`pip install python-docx`).

CÓDIGO

```
python scripts/build_contract.py build --data contratos/<cliente>/datos.json --out
"contratos/<cliente>/Contrato - <Cliente> - <YYYY-MM-DD>.docx"
python scripts/build_contract.py check --file "contratos/<cliente>/Contrato - <Cliente> - <YYYY-
MM-DD>.docx"
```

`build` ensambla el Word **sobre la plantilla corporativa** (nunca sobre un documento en blanco) y, al terminar, corre automáticamente los mismos controles que `check`. Si el script termina con un código distinto de 0, el contrato **no** debe moverse a `contratos/<cliente>/` como versión de entrega: hay que resolver lo reportado y volver a generar. La opción `--skip-check` desactiva la verificación automática y es **solo para borradores intermedios** que se sabe incompletos.

Los dos controles automáticos

Control	Falla si…
Placeholders	Queda cualquier `[PENDIENTE …]` sin resolver en el cuerpo, las tablas, el encabezado o el pie de página
Diseño corporativo	El documento perdió la referencia a la imagen de membrete del encabezado, o el bloque de firmas no tiene un párrafo con borde inferior (la línea física)

El esquema de `datos.json`

El archivo de datos es un artefacto de trabajo: vive junto al `.docx` final, como registro de lo que se generó, y sirve para reconstruir o revisar una versión posterior sin reabrir el Word.

Campo	Descripción
`titulo`	Título del contrato
`secciones[].encabezado`	Nombre de la cláusula, en el orden que corresponda
`secciones[].nivel`	Nivel de título (1 es Heading 1, 2 es Heading 2…)
`secciones[].parrafos`	Lista de textos ya redactados, un string por párrafo
`secciones[].tabla`	Opcional: `encabezados` y `filas`, para cronogramas y formas de pago
`firmantes[]`	`rol`, `nombre`, `cargo` y, opcionalmente, `identificacion` de cada parte

El diseño corporativo

`recursos/Contrato - Plantilla Base.docx` trae el membrete de Campuslands integrado como imagen de fondo del encabezado de sección: logo, franjas azul y verde superior e inferior y dirección en el pie, con los márgenes y la tipografía correctos para que el texto no invada el membrete. Por eso un contrato generado desde cero, sin ese encabezado, es **un defecto de entrega y no una opción de estilo**.

El bloque de firmas lleva **una línea horizontal** (borde inferior de párrafo o guion bajo) sobre el nombre de cada representante legal, con espacio en blanco encima. Nunca solo el nombre sin línea.

La verificación cruzada

Al terminar el borrador, se copia `wiki/plantilla-verificacion-cruzada.md` a `contratos/<cliente>/verificacion-cruzada.md` y se llena **dato por dato**, contrastando el borrador con las fuentes, no de memoria:

Dato	Valor en la fuente	Valor en el borrador	✓ / ×
Razón social del cliente			
NIT del cliente			
Representante legal del cliente			
Objeto y alcance funcional			
Exclusiones del alcance			
Valor total			
Forma de pago (hitos y porcentajes)			
Cronograma, fases y plazos			
Mantenimiento y soporte			
Garantía (duración y alcance)			

Cualquier fila con ✗ se resuelve, corrigiendo el borrador o preguntando al usuario, **antes** de correr `check`. Después, se añade una línea en `wiki/log.md` confirmando que la verificación se hizo, no solo qué fuentes se usaron.

Convenciones de nombres y de estado

Caso	Convención
Carpeta de cliente	`recursos/alcances/<cliente-slug>/`, `recursos/cotizaciones/<cliente-slug>/` y `contratos/<cliente-slug>/`
Slug	Minúsculas, sin espacios ni tildes (`casa-blanca`)
Contrato final	`contratos/<cliente-slug>/Contrato - <Cliente> - <YYYY-MM-DD>.docx`
Revisión sin firmar	`Contrato - <Cliente> - <YYYY-MM-DD> (v2).docx`, `(v3)`…; el anterior se conserva hasta que la nueva sea definitiva
Otrosí o adenda a un contrato firmado	`Otrosi 1 - <Cliente> - <YYYY-MM-DD>.docx`, `Otrosi 2 - …`; nunca se sobrescribe el original firmado

El `wiki/index.md` debe apuntar siempre a la versión vigente y usar uno de estos estados:

Estado	Significa
`Borrador`	En redacción; aún no pasó los controles de `check`
`Pendiente revisión legal`	Pasó los controles automáticos; falta la validación legal externa
`Pendiente firma`	Revisado; falta únicamente la fecha y la firma de las partes
`Firmado`	Contrato ejecutado, versión final

La memoria del agente

La wiki evita repetir preguntas y errores entre contratos:

- `wiki/datos-empresa.md` guarda los datos legales **propios** de Campuslands (razón social, domicilio, NIT y representante legal), que se repiten en todos los contratos. No se actualizan solos: antes de usarlos en un contrato nuevo se verifica con el usuario que sigan vigentes.

- `wiki/clientes/<cliente-slug>.md` guarda los datos legales confirmados de cada cliente y las condiciones particulares de sus contratos, incluidas las alertas (por ejemplo, la de un borrador cuyo alcance resultó equivocado y que quedó marcado como no vigente).
- `wiki/glosario-clausulas.md` documenta las categorías de cláusulas, los ajustes validados por proyecto y el historial del contrato madre.

Al terminar cada contrato

1. Actualizar `wiki/index.md` con la entrada del contrato y su estado.
2. Añadir una línea en `wiki/log.md` con fecha, cliente y qué se generó, confirmando la verificación cruzada y el gate.
3. Crear o actualizar `wiki/clientes/<cliente-slug>.md` con los datos legales y las condiciones confirmadas.
4. Si surgió una cláusula nueva o un ajuste reutilizable, documentarla en el glosario, previa validación del usuario.
5. Si se reemplazó el contrato madre, registrarlo en el historial de versiones.

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

La regla común

1. Un comando es una palabra precedida de `!` al inicio de un mensaje o de una línea; el texto que sigue es su objeto.
2. Es obligatorio: el agente no pide permiso ni lo reinterpreta. Solo lo limita una **regla dura** (veracidad, marca, confidencialidad, puertas de aprobación y de verificación); entonces hace lo que la regla permite y dice qué no pudo.
3. Se combinan. Si dos modos chocan, el orden es: reglas duras > `!strict-mode` y `!no-assumptions` > `!think-deep`, `!critical-mode` y `!double-check` > `!notime`.
4. Rige la tarea en curso, no las siguientes.
5. La primera línea de la respuesta nombra los comandos recibidos.
6. Un comando desconocido no se adivina: se propone el más parecido y se pregunta. `!help` lista los disponibles.
7. Si un comando cambió un entregable o la wiki, se anota en `wiki/log.md`.

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.

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:** Analiza alcances, cotización y Contrato Madre: datos duros, cifras y lo que falta antes de redactar.
`!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]`.
`!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 el checklist de datos y la estructura de cláusulas; espera confirmación antes de redactar.

Comando	Qué hace aquí
`!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 contradicciones entre el contrato, la cotización, los alcances y el Contrato Madre.
`!validate`	Comprueba, requisito por requisito, que el resultado cumple lo establecido (reglas duras, insumos, formato). Veredicto «cumple / no cumple» por ítem, con evidencia. **Aquí:** Corre la verificación cruzada (cada cifra, plazo y alcance frente a la fuente) y el gate `build_contract.py check`.
`!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**.
`!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.
`!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.
`!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 `build_contract.py` (placeholders sin resolver, membrete o firma perdidos) y repite el gate.
`!troubleshoot`	Investiga un problema técnico y determina su **causa raíz**, con evidencia. Diagnostica; no cambia nada.
`!security-check`	Revisa seguridad y confidencialidad: datos de otros clientes mezclados, credenciales, datos personales, información que no debe salir del proyecto. **Aquí:** Los datos de clientes son confidenciales: revisa que no se mezclen entre clientes y qué carpetas no deben compartirse.

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:** Revisa a la vista el `.docx` (membrete, líneas de firma, numeración, formato corporativo).
`!consistency-check`	Detecta inconsistencias entre secciones, componentes, cifras, nombres y estilos del mismo entregable.

Gestión de proyecto

Comando	Qué hace aquí
`!breakdown`	Divide un proyecto o solicitud compleja en actividades específicas, cada una con su entregable y su orden.
`!track-progress`	Informa el progreso y los pendientes (hecho, en curso, bloqueado, `[PENDIENTE]`) a partir de la bitácora y del trabajo actual.
`!risk-analysis`	Identifica riesgos, su probabilidad e impacto, y propone la mitigación de cada uno. **Aquí:** Riesgos contractuales: cláusulas de protección omitidas, ambigüedades, discrepancias de cifras o plazos.
`!requirements`	Extrae y estructura requerimientos funcionales y no funcionales de las fuentes, cada uno con su referencia de origen.
`!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:** Verificación cruzada y gate superados, contrato en su carpeta con el nombre convenido, wiki y bitácora al día.
`!agent-handoff`	Prepara el traspaso al siguiente agente del flujo comercial: resumen, entregables con su ruta, decisiones tomadas y pendientes. **Aquí:** Cierra el flujo: prepara el paquete para revisión legal y firma (contrato, verificación cruzada y pendientes).

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.
`!team-review`	Varios revisores con roles distintos evalúan por separado el mismo resultado y se consolida su evaluación. **Aquí:** Revisores con rol **jurídico (cláusulas)**, **comercial (cifras frente a la cotización)** y **forma (plantilla corporativa)**.
`!consensus`	Consolida las evaluaciones disponibles (de subagentes o de revisiones previas) en una sola recomendación, con las discrepancias a la vista.

Mantenimiento

Quiero...	Qué tocar
Cambiar una regla del agente	`CLAUDE.md` y una entrada en `CHANGELOG.md`
Cambiar el montaje o los controles	`scripts/build_contract.py` y `scripts/README.md`, más `CHANGELOG.md`
Cambiar el diseño del documento	`recursos/Contrato - Plantilla Base.docx`
Reemplazar el contrato madre	`recursos/contrato-madre/` y el historial en `wiki/glosario-clausulas.md`

Quiero...	Qué tocar
Cambiar los datos de Campuslands	`wiki/datos-empresa.md`, confirmando con el usuario
Registrar un contrato	`wiki/index.md`, `wiki/log.md` y la ficha del cliente

Problemas frecuentes

Síntoma	Causa probable	Qué hacer
`check` **falla por un** `[PENDIENTE …]`	Falta un dato en las fuentes	Resolver el dato con el usuario o la fuente, regenerar con `build`; no entregar con placeholders
`check` **falla por el membrete**	El Word no se montó sobre la plantilla, o se editó a mano	Reconstruir con `build` en lugar de editar el documento
`check` **falla por la línea de firma**	El bloque de firmas perdió su borde	Volver a generar el bloque con `build`
Una cláusula admite dos lecturas	Alcance o cotización ambiguos	Detenerse y preguntar al usuario antes de redactar
Cambió algo tras la firma	Alcance o valor modificados	Preparar un otrosí numerado; nunca sobrescribir el contrato firmado
El contrato madre cambió	Se reemplazó la plantilla de contenido	Registrar el cambio en el historial y revisar si los contratos generados quedaron desactualizados
Dudas sobre subir el repo	Contiene NIT, representantes y cifras	Confirmar con el usuario qué carpetas excluir antes de compartirlo

Referencia rápida de archivos

Archivo	Para qué lo abres
`CLAUDE.md`	Reglas duras, flujo y convención de nombres
`scripts/build_contract.py` y `scripts/README.md`	El montaje, los controles y el esquema de `datos.json`
`wiki/glosario-clausulas.md`	Categorías de cláusulas, ajustes validados e historial del contrato madre
`wiki/datos-empresa.md`	Datos legales de Campuslands
`wiki/clientes/<cliente>.md`	Datos y condiciones de cada cliente
`wiki/plantilla-verificacion-cruzada.md`	La tabla de verificación
`wiki/index.md`	Estado de cada contrato
`CHANGELOG.md`	Historia de los cambios al agente