MANUALE OPERATIVO

Claude

la guida completa

Dal primo prompt all'automazione:
Claude Code, Cowork, le skill e l'API

Gian Angelo Geminiani

Edizione 2026 · Rev. 3 · Luglio 2026

Note legali

Claude: la guida completa — Edizione 2026 · Rev. 3 · Luglio 2026

Pubblicazione **indipendente e non ufficiale**. Questo libro non è prodotto, autorizzato, sponsorizzato, approvato né altrimenti collegato ad Anthropic PBC o alle sue affiliate. Le opinioni e le interpretazioni sono dell'autore.

Marchi. "Claude", "Anthropic", "Claude Code", "Cowork", "Claude Design" e gli altri nomi di prodotto, servizio, logo e marchio citati appartengono ai rispettivi titolari. Sono usati esclusivamente in funzione **identificativa e descrittiva** (uso nominativo / fair use), per riferirsi ai prodotti oggetto del manuale, e non implicano alcun rapporto di sponsorizzazione o approvazione. Nessun logo o marchio figurativo di terze parti è riprodotto in questo volume.

Accuratezza dei dati. Le informazioni su versioni, comandi, prezzi, menu e funzionalità si riferiscono a prodotti di terze parti in rapida evoluzione: sono fornite **"così come sono" ("as is")**, verificate alla data indicata in ciascun capitolo e **soggette a modifica** senza preavviso. Per i valori aggiornati fai riferimento alle fonti ufficiali (claude.com, support.claude.com, code.claude.com, platform.claude.com) e al repository companion.

Indice

F.1 — Prefazione

Front matter — Livello 0. Dati di prodotto verificati il 22/06/2026 su fonti ufficiali.

Obiettivo

Questa prefazione spiega per chi è il libro, come è organizzato e come leggerlo. Al termine saprai a quale livello partire e come interpretare i box, i tag e le figure che incontrerai nei capitoli.

A chi si rivolge

Il libro è per chi vuole padroneggiare Claude in italiano, dal primo messaggio fino alle attività avanzate. Non dà per scontata alcuna competenza tecnica nei primi livelli, e alza l'asticella mano a mano. Tre profili tipici:

- chi parte da zero e vuole usare bene la chat;
- chi installa e configura Claude in locale (Code, Cowork);
- chi automatizza, integra e lavora in team.

Non devi leggere tutto in ordine. Il capitolo F.4 propone tre percorsi di lettura a seconda di cosa ti serve.

Cosa il libro **non** è: un corso di programmazione generica, una rassegna di prodotti concorrenti o un listino prezzi. Quando un dato è soggetto a cambiare (prezzi, versioni), il libro te lo segnala e ti rimanda alla fonte aggiornata, invece di fingere una stabilità che non c'è.

Claude non è un solo prodotto

Il primo equivoco da sciogliere: "Claude" non è soltanto la chat nel browser. È un ecosistema. C'è la chat, c'è lo strumento per programmare (Claude Code), c'è il lavoro agentico sul tuo computer (Cowork), c'è la parte di design, e ci sono le API per chi sviluppa. Capire come questi pezzi si parlano è metà del valore del libro: il capitolo F.2 ne dà la mappa.

Come è strutturato

Il contenuto è diviso in livelli di complessità crescente. Ogni capitolo dichiara il proprio livello, così sai cosa aspettarti:

- **Livello 1 — Fondamenti:** usare bene la chat.
- **Livello 2 — Installazione locale:** Desktop e Code.

- **Livello 3 — Lavoro quotidiano:** Cowork, Projects, documenti.
- **Livello 4 — Design:** dal canvas al codice.
- **Livello 5 — Skills e identità:** dare a Claude la tua voce.
- **Livello 6 — Avanzato:** automazioni, integrazioni, API.

Prima dei livelli c'è il front matter (questa parte); alla fine, un progetto completo e le appendici.

Come usare box, tag e figure

Per leggere senza sorprese, tieni a mente tre convenzioni.

I **box** segnalano informazioni fuori dal flusso: *Nota* (un dettaglio utile), *Attenzione* (un errore da evitare), *Tip* (una scorciatoia).

I **tag** dicono quanto è stabile un'informazione. *(EVERGREEN)* indica concetti che cambiano di rado. *(VOLATILE)* indica dati che possono cambiare presto: versioni, comandi, prezzi, nomi di menu.

Le **figure** sono numerate per capitolo e hanno sempre una breve didascalia e un testo alternativo, così restano comprensibili anche a chi usa lettori di schermo o legge una versione senza immagini. L'identificativo unisce il capitolo e la progressione: la prima figura del cap. L2.1 è la Figura L2.1.1, la prima tabella del front matter F.2 è la Tabella F.2.1.

Una nota onesta sul prodotto che cambia

Claude evolve in fretta: nuovi modelli, nuove funzioni, menu che si spostano. Un libro di carta rischia di nascere vecchio. Per questo i dati volatili sono marcati come tali e raccolti, dove serve, in un **repo companion** online che accompagna il libro con comandi aggiornati ed errata. Quando vedi un dato *(VOLATILE)*, considera la data di verifica indicata e controlla la fonte ufficiale se ti serve il valore più recente.

> **Nota:** ogni capitolo riporta in alto la data in cui i dati di prodotto sono stati verificati. È il tuo riferimento per capire quanto “fresca” è un’informazione.

Riepilogo

1. Il libro va dal principiante all’esperto: parti dal livello adatto a te.
2. Claude non è solo la chat, ma un **ecosistema** di prodotti (cap. F.2).
3. Il contenuto è diviso in **sei livelli** di complessità crescente.
4. **Box**, **tag** *(EVERGREEN)*/*(VOLATILE)* e **figure numerate** ti orientano nella lettura.
5. I dati che cambiano sono datati e raccolti nel **repo companion**: verifica sempre alla fonte quando ti serve il valore corrente.

Prossimo passo

Nel **cap. F.2 — L’ecosistema Claude** vediamo la mappa completa dei prodotti: cosa fa ciascuno, dove gira e quando conviene usarlo rispetto agli altri.

F.2 — L'ecosistema Claude

Front matter — Livello 0. Dati di prodotto verificati il 02/07/2026 su fonti ufficiali.

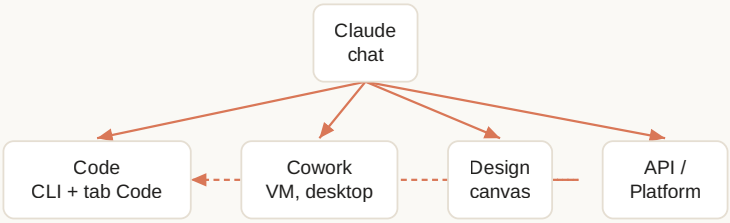
Obiettivo

Al termine avrai una mappa mentale dei prodotti Claude: cosa fa ciascuno, dove gira e quando conviene usarlo rispetto agli altri. È la bussola che orienta tutto il resto del libro.

Un'unica intelligenza, molte porte d'accesso

Sotto i vari prodotti c'è lo stesso motore: i modelli Claude. Cambia il modo in cui ci accedi e cosa puoi fargli fare. La chat è la porta principale; le altre porte aggiungono autonomia (Cowork), integrazione col codice (Code), lavoro visuale (Design) o controllo programmatico (API).

Figura F.2.1 — Le principali porte d'accesso a Claude. Alt-text: diagramma verticale con la chat in alto che si dirama in Code, Cowork, Design e API; da Design parte un handoff verso Code.



I prodotti, uno per uno

Claude (chat). L'assistente conversazionale su web, app desktop e mobile. È il punto di partenza per domande, scrittura, analisi e generazione di file. Disponibile a tutti i piani.

Claude Code. Lo strumento di coding agentico. Vive come comando da terminale (CLI) e come tab **Code** nell'app desktop, sullo stesso motore. Serve a chi sviluppa software. Richiede un piano a pagamento.

Claude Cowork. Porta le capacità agentiche nell'app desktop per lavoro non di programmazione: ricerche, documenti, organizzazione di file. Gira in una VM isolata (macchina virtuale separata dal sistema) e tocca solo le cartelle che colleghi. Solo desktop (macOS/Windows), piani a pagamento.

Claude Design. Un canvas per creare UI, prototipi e presentazioni conversando con Claude. Si integra con Code: quando un design è pronto, lo "passi" allo sviluppo invece di ripartire da uno screenshot. Beta, a pagamento.

Claude API / Developer Platform. L'accesso programmatico ai modelli, con SDK e Console. Serve a chi integra Claude nei propri software. È pay-as-you-go.

Add-in e browser. Claude vive anche dentro Excel/Word/PowerPoint (beta) e come estensione per Chrome che agisce sulle pagine web (beta, a pagamento).

I più recenti (beta). L'ecosistema continua a crescere. **@Claude (Claude Tag)** porta Claude dentro Slack come un membro del team: chiun-
que nel canale lo tagga e gli delega un compito (Team/Enterprise). **Claude Science** è un workbench (banco di lavoro) per la ricerca scientifica, con decine di database e toolkit integrati; è anche il primo prodotto desktop disponibile su Linux oltre che su macOS (piani a pagamento). Sono in beta: aspettati cambiamenti.

Quando usare cosa

La domanda giusta non è "quale prodotto è migliore", ma "qual è il più adatto a questo compito". La tabella riassume la scelta.

Tabella F.2.1 — Prodotto, dove gira, quando conviene.

Prodotto	Dove gira	Quando usarlo
Chat	web/desktop/mobile	domande, scrittura, analisi
Code	terminale + desktop	sviluppo software
Cowork	desktop (VM)	task lunghi sui tuoi file
Design	web/desktop	UI, prototipi, slide
API	server/codice	integrare Claude nei tuoi software

> **Nota:** chat, Cowork e Code convivono nella stessa app desktop come tre tab. Cambiare tab significa cambiare modo di lavorare, non applicazione.

Come si parlano tra loro

Il valore vero nasce quando i prodotti si passano il lavoro. Da Design "passi" un'interfaccia a Code, che la costruisce davvero invece di ripartire da zero. Cowork e Code condividono la stessa cartella di progetto e le stesse istruzioni. Una Skill scritta una volta (Livello 5) guida Claude allo stesso modo in chat, in Cowork e in Code. Imparare i singoli prodotti è utile; imparare a farli collaborare è ciò che rende Claude un ecosistema e non una somma di strumenti.

Riepilogo

1. Sotto a tutto ci sono gli stessi modelli Claude: cambia la porta d'accesso.

2. La **chat** è il punto di partenza generale.
3. **Code** è per programmare; **Cowork** per task autonomi sui tuoi file.
4. **Design** copre la parte visuale e passa il lavoro a Code.
5. Le **API** servono a integrare Claude nei propri software.

Prossimo passo

Nel **cap. F.3 — Modelli e piani** scegliamo il modello e il piano giusti: quali differenze contano davvero e cosa sblocca ciascun abbonamento.

F.3 — Modelli e piani

Front matter — Livello 0. Dati di prodotto verificati il 02/07/2026 su fonti ufficiali.

Obiettivo

Al termine saprai distinguere le famiglie di modelli Claude e capire cosa sblocca ciascun piano, così da scegliere la combinazione giusta per te. I dati puntuali qui sono volatili: contano i criteri, più dei numeri.

Le tre famiglie di modelli

Claude offre tre famiglie principali, pensate per bilanci diversi tra intelligenza, velocità e costo. Il nome resta (Opus, Sonnet, Haiku), cambia il numero di versione nel tempo.

Tabella F.3.1 — Le famiglie e la versione corrente.

Famiglia	Versione	Quando usarla
Opus	4.8	ragionamento complesso
Sonnet	5	uso quotidiano
Haiku	4.5	risposte rapide

In breve: **Opus** è il più capace, adatto a compiti difficili e al lavoro agentico lungo; **Sonnet** è l'equilibrio ideale tra resa e velocità, la scelta "tutti i giorni" (ed è il modello proposto di default sui piani Free e Pro); **Haiku** è il più rapido, perfetto per risposte brevi e task semplici.

Sopra Opus c'è una classe di modelli più potenti (i "Mythos"), pensata per il ragionamento e il lavoro agentico più estremi: **Fable 5** (versione resa disponibile più ampiamente) e **Mythos 5** (ad accesso ristretto). La loro storia recente spiega da sola perché in questo libro ragioniamo per

famiglie: lanciati a giugno 2026, sono stati **sospesi dopo tre giorni** per una direttiva governativa, poi **riattivati** tre settimane dopo, alla revoca della direttiva. E il rientro ha condizioni sue: Fable 5 è incluso nei piani a pagamento solo in parte e per un periodo limitato, dopodiché si usa tramite crediti d'uso (cap. L6.4); ha protezioni più strette degli altri modelli, che a volte bloccano richieste legittime; e i suoi dati seguono regole di conservazione diverse. Morale: i modelli di punta vanno e vengono, e ognuno arriva con le proprie regole. Per lo stato corrente fai riferimento al repo companion e alle fonti ufficiali.

> **Attenzione:** nomi, stringhe e perfino la *disponibilità* dei modelli cambiano di frequente. Non memorizzare un modello come "quello giusto per sempre": verifica la versione corrente quando ti serve.

Come scegliere il modello

Una regola pratica che resiste ai cambi di versione: parti da Sonnet. Se la risposta non regge la complessità del compito, sali a Opus. Se ti serve solo velocità su qualcosa di semplice, scendi a Haiku. Non scegliere il modello più potente "per sicurezza": spesso paghi di più senza vantaggi.

I piani

Il piano determina a quali prodotti accedi e quanto puoi usarli. Oltre alla chat, ogni livello aggiunge qualcosa.

Tabella F.3.2 — Cosa sblocca ogni piano.

Piano	Per chi	Cosa aggiunge
Free	provare	Web search, file, 5 Projects
Pro	singolo	Code, Cowork, Design, Research
Max	uso intenso	usage 5x/20x, priorità
Team	gruppi 5-150	admin, SSO, deploy
Enterprise	aziende	controlli e sicurezza

Il piano **Free** copre bene la chat, la ricerca web, la creazione di file, le Skills e fino a cinque Projects, ma non include Code, Cowork né Design. **Pro** è lo spartiacque: sblocca proprio quei tre. **Max** non aggiunge prodotti ma alza di molto i limiti d'uso. **Team** ed **Enterprise** aggiungono amministrazione, sicurezza e gestione centralizzata per le organizzazioni.

> **Nota:** i prezzi indicativi (in USD) erano, a giugno 2026, circa 17-20 $/mese per Pro e da ~100 $/mese per Max. Sono dati volatili: controlla il valore aggiornato su claude.com/pricing.

Errori comuni

- **Scegliere Free per usare Code o Cowork.** Non bastano: servono almeno Pro.
- **Pagare Max senza averne bisogno.** Max serve per i limiti d'uso, non per sbloccare prodotti nuovi: se non tocchi i limiti di Pro, non ti serve.
- **Fissare un nome di modello.** Le versioni cambiano; ragiona per famiglia.

Riepilogo

1. Tre famiglie: **Opus** (potenza), **Sonnet** (equilibrio), **Haiku** (velocità).
2. Parti da Sonnet; sali a Opus se serve, scendi a Haiku per i task semplici.
3. **Free** non include Code, Cowork e Design; **Pro** sì.
4. **Max** alza i limiti d’uso; **Team/Enterprise** aggiungono gestione e sicurezza.
5. Prezzi e versioni sono volatili: verifica su fonti ufficiali.

Prossimo passo

Nel **cap. F.4 — Percorsi di lettura** scegli da dove iniziare il libro in base a cosa ti serve, con tre itinerari pronti.

F.4 — Percorsi di lettura

Front matter — Livello 0. Dati di prodotto verificati il 22/06/2026 su fonti ufficiali.

Obiettivo

Al termine saprai da dove iniziare e quali capitoli seguire in base al tuo obiettivo. Il libro copre molto: questo capitolo ti evita di leggere tutto in ordine quando non serve.

Leggere a salti, non per forza in fila

I capitoli sono ordinati per livello crescente, ma non sei obbligato a seguirli linearmente. Chi non programma può saltare i livelli di sviluppo; chi sviluppa può attraversare in fretta i fondamenti. Qui sotto trovi tre itinerari pronti, ognuno pensato per un obiettivo diverso. Tutti partono dal front matter (F) e chiudono con il progetto finale (C.1), così vedi un caso completo.

I tre percorsi

Tabella F.4.1 — Tre itinerari a seconda dell'obiettivo.

Percorso	Per chi	Itinerario
Solo Cowork	non-tecnici	F → L1 → L3 → L5 → C.1
Code/Design	chi sviluppa	F → L2 → L4 → L6 → C.1
Da zero	primo progetto	F → L1 → L2 → L4 → C.1

Solo Cowork. Per chi vuole far lavorare Claude sui propri file senza toccare il terminale. Dopo i fondamenti della chat (L1), passa al lavoro quotidiano con Cowork e Projects (L3), poi alle Skills per dare a Claude la tua voce (L5).

Solo Code/Design. Per chi sviluppa o disegna interfacce. Installa gli strumenti locali (L2), lavora con Design e il ponte verso il codice (L4), poi spinge su automazioni e integrazioni avanzate (L6).

Da zero al primo progetto. Per chi parte da zero e vuole arrivare a un risultato concreto. Fondamenti (L1), installazione (L2) e la parte di Design utile a costruire qualcosa di reale (L4).

> **Tip:** se non sai dove collocarti, segui “Da zero”: tocca i punti essenziali di ogni area e ti lascia liberi di approfondire dopo.

La legenda delle icone

Nei capitoli, un’icona a margine segnala a quale percorso quel contenuto è più utile. Servono a orientarti con un colpo d’occhio.

- **Cowork:** contenuto chiave per il percorso non-tecnico.
- **Code/Design:** contenuto chiave per chi sviluppa o disegna.
- **Da zero:** tappa consigliata per chi parte da principiante.

Un capitolo può avere più icone: significa che è utile a più percorsi. L’assenza di icone indica materiale trasversale, valido per tutti.

Saltare senza perdersi

Ogni capitolo si apre con una sezione **Prerequisiti**: ti dice cosa dare per acquisito e a quali capitoli tornare se manca un pezzo. È la rete di sicurezza che ti permette di saltare. Se entri in un capitolo e i prerequisiti ti sono chiari, procedi; altrimenti recuperi solo ciò che serve, senza rileggere tutto.

Allo stesso modo, i rimandi “vedi cap. X” non sono obblighi: segnalano dove trovare un approfondimento se in quel momento ti serve. Puoi ignorarli e tornarci più tardi.

Dove si arriva

Tutti i percorsi convergono sulla **chiusura** del libro. Il **cap. C.1** è un progetto end-to-end che attraversa i livelli su un caso reale: lì vedi i pezzi lavorare insieme. Le **appendici** (cap. C.2) raccolgono glossario, troubleshooting e link ufficiali, da consultare quando servono più che da leggere in sequenza.

Riepilogo

1. I capitoli sono per livelli, ma puoi leggerli a salti.
2. Tre percorsi: **Solo Cowork**, **Code/Design**, **Da zero**.
3. Tutti partono da F e chiudono con il progetto finale C.1.
4. Un’icona a margine indica a quale percorso serve di più un contenuto.
5. Nel dubbio, segui “Da zero”.

Prossimo passo

Inizia il **Livello 1** con il **cap. L1.1 — Primo contatto**: apri Claude, manda il primo messaggio e impara a leggere una risposta.

Capitolo L1.1 — Primo contatto

Livello 1 — Fondamenti. Dati di prodotto verificati il 22/06/2026 su fonti ufficiali.

Obiettivo

Al termine avrai aperto Claude sulla piattaforma che preferisci, mandato il tuo primo messaggio, allegato un file e saprai leggere una risposta con occhio critico. È la base su cui poggia tutto il resto.

Prerequisiti

- Un account Claude (vedi cap. F.3). Per iniziare basta il piano Free.
- Una connessione internet attiva.

Dove trovi Claude

Claude è disponibile su tre superfici, sincronizzate quando accedi con lo stesso account:

- **Web:** vai su claude.ai dal browser. È il modo più veloce per provarlo.
- **Desktop:** l'app per macOS e Windows (vedi cap. L2.1 per installarla).
- **Mobile:** le app per iOS e Android, utili per continuare in mobilità.

Le conversazioni, i progetti e le preferenze ti seguono tra i dispositivi. Puoi iniziare una chat sul telefono e riprenderla dal computer.

Quale scegliere? Per provare subito, il **web** non richiede installazione: basta un browser. L'app **desktop** conviene quando usi Claude ogni giorno o ti servono le funzioni locali come Cowork e Code (Livello 2). Il **mobile** è comodo per catturare un'idea al volo o continuare in mobilità.

Non è una scelta definitiva: l'account è lo stesso e passi dall'uno all'altro quando vuoi.

> **Nota:** un'eccezione alla sincronizzazione sono le chat in incognito (icona fantasma): non vengono salvate e quindi non compaiono sugli altri dispositivi. Sono pensate per domande usa-e-getta.

Il primo messaggio

L'uso è diretto: scrivi nella casella di testo e invii. Non serve un comando speciale né una sintassi particolare. Tratta Claude come un collaboratore competente ma senza contesto sul tuo lavoro: più sei chiaro, migliore è la risposta. Approfondiamo come scrivere buoni messaggi nel cap. L1.2.

Un primo esperimento utile: chiedi qualcosa di concreto, ad esempio "Riassumi questo testo in cinque punti" incollando un paragrafo. Vedrai subito come Claude struttura la risposta.

Un esempio concreto

Immagina di incollare il testo di una mail ricevuta e di scrivere: «Riassumi questa mail in tre punti e suggerisci una risposta cortese». Claude risponde con i tre punti e una bozza pronta da adattare. Da qui puoi continuare nello stesso filo: «Rendi la risposta più formale» oppure «Aggiungi una proposta di data per la call». Non riparti da zero: finché resti nella stessa chat, Claude tiene il contesto dei messaggi precedenti.

È questa la dinamica di fondo: una conversazione, non un singolo comando. Ogni messaggio si appoggia ai precedenti, ed è ciò che rende la chat più utile di una ricerca secca.

> **Tip:** quando un risultato ti soddisfa solo in parte, correggi il tiro con un messaggio breve invece di riscrivere tutta la richiesta. Spesso basta una riga (“più corto”, “tono informale”) per arrivare dove vuoi.

L’interfaccia in breve

L’interfaccia è essenziale. Attorno alla casella di testo trovi pochi comandi che vale la pena riconoscere subito:

- Il pulsante **“+”** apre gli allegati (file e foto).
- Il menu **“Search and tools”** (icona a cursori) raccoglie strumenti e stili, come Web search (ricerca web) e lo stile di risposta.
- L’icona a forma di **fantasma** avvia una chat in incognito: una conversazione temporanea, non salvata nello storico.

> **Nota:** la posizione esatta dei pulsanti può cambiare tra versioni e tra web, desktop e mobile. I nomi e le funzioni, però, restano gli stessi.

Allegare file e immagini

Puoi dare a Claude documenti e immagini da analizzare. I formati supportati includono PDF, DOCX, CSV, TXT, HTML, JSON e altri; per le immagini, JPEG, PNG, GIF e WebP. I file Excel (XLSX) richiedono che sia attiva Code execution (esecuzione del codice; vedi cap. L1.3).

Tabella L1.1.1 — Modi per allegare e limiti tipici.

Come	Cosa	Limite indicativo
Pulsante “+”	file e foto	~500 MB/file
Trascina e rilascia	più file	~20 file/chat
Incolla	immagini	dagli appunti

> **Attenzione:** dai documenti che non sono PDF, Claude estrae di norma solo il testo, non le immagini incorporate. Per i PDF, l’analisi visiva è più completa sotto le ~100 pagine. I limiti sono volatili: verificali se carichi file grandi.

Cosa Claude fa con gli allegati

Allegare un file non significa solo «darlo in pasto» a Claude: puoi chiedergli di estrarne i dati, riassumerlo, confrontarlo con un altro documento o tradurlo. Più sei preciso su cosa vuoi che faccia, più la risposta è centrata. «Estrai dalla tabella i totali per mese» è meglio di «guarda questo file». Se un documento è lungo, indica la parte che conta: aiuti Claude a concentrarsi e riduci le risposte generiche.

Leggere una risposta con criterio

Claude è utile, ma non infallibile. Tre abitudini sane fin dal primo giorno:

- **Controlla i fatti che contano.** Per date, numeri e nomi, chiedi la fonte o verifica altrove. Su informazioni recenti, attiva Web search (cap. L1.3).
- **Chiedi di mostrare il ragionamento.** “Spiega come ci sei arrivato” rende la risposta più trasparente e più facile da correggere.
- **Itera.** La prima risposta è una bozza. Affinala con un secondo messaggio invece di rifare tutto da capo.

- **Diffida delle risposte troppo sicure su dettagli verificabili.** Un modello linguistico può esporre con sicurezza un dato sbagliato (è il fenomeno delle «allucinazioni»): il tono sicuro non garantisce la correttezza.

Non è un difetto da temere, ma un modo d'uso da imparare. Claude dà il meglio quando ragiona, redige e trasforma il materiale che gli fornisci, più che quando gli chiedi di ricordare fatti "a memoria". Per i dati che cambiano nel tempo, Web search è il complemento naturale: porta informazioni aggiornate con le fonti.

In pratica: la tua prima conversazione

1. Apri claude.ai (o l'app) e accedi.
2. Scrivi una richiesta concreta, ad esempio un riassunto o una bozza di email.
3. Allega un file con il pulsante "+" e chiedi a Claude di usarlo.
4. Leggi la risposta e mandane una seconda per migliorarla.
5. Prova la chat in incognito (icona fantasma) per una domanda usa-e-getta.
6. Ripeti la stessa richiesta su un'altra superficie (web o mobile) e nota che la conversazione ti segue.

Errori comuni

- **Richiesta troppo vaga.** "Scrivimi qualcosa" produce risposte generiche. Indica scopo, destinatario e formato (cap. L1.2).
- **File non letto come ti aspetti.** Se è un documento con immagini, ricorda che Claude ne estrae il testo; descrivi a parole ciò che conta.
- **Fidarsi al 100%.** Verifica i dati sensibili; non dare per certa una data o un numero senza controllo.

- **Usare l’incognito per lavoro che vuoi ritrovare.** Quelle chat non si salvano: se ti serve riprenderle, usa una chat normale.

Riepilogo

1. Claude è su **web, desktop e mobile**, sincronizzato col tuo account.
2. Si usa scrivendo nella casella di testo: niente sintassi speciale.
3. Riconosci **“+”** (allegati), **Search and tools** (strumenti/stili) e l’icona **fantasma** (incognito).
4. Puoi allegare molti formati; attento ai limiti e all’estrazione solo testo.
5. Leggi le risposte con criterio: verifica, chiedi il ragionamento, itera.

Prossimo passo

Nel **cap. L1.2 — Conversare bene** vediamo come scrivere prompt efficaci, con esempi prima/dopo e gli errori più comuni da evitare.

Capitolo L1.2 — Conversare bene

Livello 1 — Fondamenti. Dati di prodotto verificati il 22/06/2026 su fonti ufficiali.

Obiettivo

Al termine saprai scrivere richieste (prompt) che ottengono risposte utili al primo colpo: chiare, con il contesto giusto e nel formato che ti serve. È l'abilità che fa la differenza tra usare Claude e usarlo bene.

Prerequisiti

- Aver mandato il primo messaggio (cap. L1.1).

La regola d'oro

Pensa a Claude come a un nuovo collaboratore: bravo, ma senza contesto sulle tue abitudini. Se la tua richiesta confonderebbe una persona competente che non conosce il tuo lavoro, confonderà anche Claude. Da qui derivano cinque principi pratici.

Nessuno di questi principi richiede tecnicismi: sono lo stesso buon senso che useresti delegando un compito a una persona. La differenza è che con Claude il costo di riprovare è quasi nullo, quindi puoi permetterti di essere esplicito senza timore di «chiedere troppo».

1. Sii chiaro e specifico

Di' esplicitamente cosa vuoi, per chi e con quali vincoli. Indica lunghezza, tono e formato dell'output. Quando l'ordine conta, usa passi numerati. "Scrivi qualcosa sul prodotto" lascia troppo spazio; "Scrivi tre righe

di descrizione per il prodotto X, tono professionale, per un pubblico tecnico” guida la risposta.

2. Dai contesto e motivazione

Spiegare il perché di una richiesta aiuta Claude a centrarla. “Riassumi questo report **per un dirigente che ha due minuti**” porta a un risultato diverso da un riassunto generico. Il contesto non è un di più: è ciò che orienta le scelte. Anche un vincolo, se posto in positivo, aiuta: invece di «non essere tecnico», scrivi «usa parole che capirebbe un cliente non del settore».

3. Usa esempi

Mostrare uno o due esempi del risultato che vuoi è uno dei modi più affidabili per guidare formato e tono. Se ti serve una scheda prodotto con una certa struttura, incolla una scheda già fatta come modello. Tre-cinque esempi ben scelti valgono più di mille spiegazioni.

4. Controlla il formato

Di’ cosa vuoi, non cosa non vuoi. “Rispondi con un elenco puntato di massimo cinque voci” è meglio di “non essere prolisso”. Se vuoi una tabella, chiedila; se vuoi JSON o Markdown, specificalo. Dichiarare il formato fa risparmiare giri: se ti serve una tabella a tre colonne o un elenco numerato, dillo subito invece di riformattare dopo.

5. Itera

La prima risposta è una bozza. Invece di ricominciare, correggi il tiro: “Più breve”, “Tono più informale”, “Aggiungi un esempio concreto”. Affinare in due o tre passaggi è più rapido che cercare il prompt perfetto al primo tentativo.

Dare ruolo, struttura e vincoli

Oltre ai cinque principi, due mosse alzano la qualità sulle richieste complesse.

Assegna un ruolo. Indicare il punto di vista da cui rispondere orienta tono e profondità: «Rispondi come un revisore di bilancio» o «Spiega come faresti a un collega non tecnico». Non è un trucco scenico: restringe il campo delle risposte plausibili a quelle che ti servono davvero.

Struttura le richieste lunghe. Quando una richiesta ha molte parti, separale: un blocco per il contesto, uno per il compito, uno per il formato voluto. Anche solo usare trattini o titoletti aiuta Claude a non perdere pezzi. Una richiesta ordinata produce una risposta ordinata.

Prima e dopo

La tabella mostra come passare da una richiesta debole a una efficace. Il principio è sempre lo stesso: aggiungere scopo, destinatario e formato.

Tabella L1.2.1 — Stesso intento, prompt debole e prompt migliore.

Intento	Prompt debole	Prompt migliore
Email	“scrivi un’email”	“email breve, cliente, tono cortese”
Riassunto	“riassumi”	“5 punti per un dirigente di fretta”
Codice	“fai una funzione”	“funzione che valida un’email, con test”

> **Tip:** se non sai da dove partire, scrivi prima l’obiettivo (“voglio ottenere…”), poi il contesto, infine il formato. Tre righe bastano.

Un esempio completo

Mettiamo insieme i principi su un caso reale. Parti da una richiesta d'istinto:

> Scrivimi un post per annunciare il nuovo prodotto.

È vaga: niente pubblico, niente canale, niente tono. Ecco la stessa richiesta riscritta applicando scopo, destinatario e formato:

> Scrivi un post LinkedIn (max 120 parole) per annunciare il lancio del prodotto X a professionisti del settore. Tono competente ma caldo, senza superlativi. Chiudi con una domanda che inviti al commento.

La seconda versione dichiara scopo, pubblico, canale, lunghezza, tono e perfino la chiusura. A Claude resta poco da indovinare, e la prima risposta è già vicina a ciò che volevi: il tempo speso a scrivere bene la richiesta lo recuperi sulle correzioni che non dovrai fare.

Iterare o ripartire?

Iterare nella stessa chat conserva il contesto: è la scelta giusta quando rifini un risultato. Conviene invece aprire una **nuova chat** quando cambi argomento del tutto, così il contesto vecchio non «sporca» le risposte nuove. Regola pratica: stesso obiettivo, stessa chat; obiettivo diverso, chat nuova.

Spezzare un compito grande

Le richieste enormi («scrivimi il piano marketing completo») producono risposte generiche, perché Claude deve indovinare troppo. Conviene procedere a tappe: prima l'indice, poi una sezione alla volta, infine la

revisione d'insieme. A ogni passo dai un obiettivo chiaro e controlli il risultato prima di andare avanti. Così mantieni il controllo e ottieni una qualità costante, invece di un muro di testo da sistemare tutto in una volta. Vale per testi, analisi e codice allo stesso modo: è la differenza tra delegare un progetto e delegare un passo.

Lungo non vuol dire migliore

Essere specifici non significa scrivere paragrafi. Spesso tre righe ben mirate — obiettivo, contesto, formato — battono mezza pagina di istruzioni confuse. Punta alla chiarezza, non alla quantità: ogni parola che non aggiunge un vincolo è rumore. Se ti accorgi di ripeterti, togli invece di aggiungere. Un buon prompt si riconosce così: nessuno dei suoi pezzi è eliminabile senza perdere qualcosa.

Errori comuni

- **Richiesta generica.** Senza scopo e destinatario, la risposta è anonima.
- **Troppi obiettivi in un colpo.** Spezza i compiti complessi in passi.
- **Dire solo cosa non vuoi.** Indica il risultato desiderato, non i divieti.
- **Non dare esempi.** Quando il formato conta, un modello vale più di una descrizione.
- **Arrendersi alla prima risposta.** Itera: è parte del metodo, non un fallimento.

In pratica: trasforma un prompt debole

1. Parti da una richiesta vaga che useresti d'istinto.
2. Aggiungi **scopo** ("serve per…") e **destinatario** ("per…").
3. Specifica il **formato** (lunghezza, struttura, tono).
4. Se hai un esempio del risultato voluto, incollalo.

5. Manda, valuta e **itera** con una correzione mirata.

Riepilogo

1. Tratta Claude come un collega competente ma senza il tuo contesto.
2. Sii **chiaro e specifico**: scopo, destinatario, vincoli.
3. Dai **contesto** ed **esempi**; guidano più di mille parole.
4. Controlla il **formato** dicendo cosa vuoi, non cosa eviti.
5. **Itera**: la prima risposta è una bozza da affinare.

Prossimo passo

Nel **cap. L1.3 — Impostazioni e stili** configuriamo tono, preferenze e capacità (Web search, Code execution, Memory) per non doverle ripetere a ogni chat.

Capitolo L1.3 — Impostazioni e stili

Livello 1 — Fondamenti. Dati di prodotto verificati il 22/06/2026 su fonti ufficiali.

Obiettivo

Al termine saprai configurare Claude perché risponda nel tuo tono e con le capacità giuste, senza ripetere le stesse istruzioni a ogni chat. Vedremo le preferenze d'account, gli Styles (stili) e le capacità Web search (ricerca web), Code execution (esecuzione del codice) e Memory (memoria).

Prerequisiti

- Saper avviare una conversazione (cap. L1.1).

Istruzioni a livello di account

In **Settings → Instructions for Claude** imposti preferenze che valgono per tutte le conversazioni: come ti chiami, in che lingua scrivere, il tono che preferisci, termini ricorrenti del tuo lavoro. È il posto giusto per le cose che diresti a ogni chat: scriverle una volta sola ti fa risparmiare tempo.

È diverso da un'istruzione data nella singola chat, che vale solo lì: queste valgono ovunque. Esempi utili da mettere qui: «scrivi sempre in italiano», «preferisco risposte concise con esempi», «sono uno sviluppatore», «non usare emoji». Tienile brevi e aggiornale quando le tue esigenze cambiano: è il livello più comodo per le preferenze stabili.

Stili di risposta

Gli **Styles** controllano *come* Claude comunica. Sono disponibili alcuni preset — Normal, Concise (conciso), Formal (formale), Explanatory (esplicativo) — e puoi crearne di personalizzati, anche caricando esempi della tua scrittura. Li trovi nel menu “Search and tools” accanto alla casella di testo.

Quando usarli? **Concise** per risposte rapide e dense; **Explanatory** quando stai imparando e vuoi più contesto; **Formal** per testi professionali. Lo stile **custom** è il più potente: carichi alcuni tuoi testi e Claude ne imita registro e ritmo. È il primo passo verso il «far suonare Claude come te», che approfondiamo con le Skills nel Livello 5.

> **Attenzione:** gli Styles stanno evolvendo verso le Skills (vedi Livello 5). Funzioni e nomi potrebbero cambiare: è un’area volatile, verifica lo stato attuale nelle impostazioni.

Le capacità da conoscere

Oltre al tono, puoi attivare capacità che ampliano ciò che Claude può fare. La tabella le riassume; sotto, il dettaglio.

Tabella L1.3.1 — Capacità principali e disponibilità.

Funzione	A cosa serve	Disponibilità
Web search	dati aggiornati	tutti i piani
Code execution	creare file	tutti i piani
Memory	ricorda lo storico	tutti i piani
Chat search	cerca chat passate	a pagamento

Web search. Attiva la ricerca live sul web, con citazioni delle fonti. È indispensabile per informazioni recenti, dove la conoscenza “a memo-

ria” del modello non basta. Si attiva dal pulsante “+” o dal menu strumenti. Consuma i tuoi limiti d’uso. Su Team ed Enterprise va abilitata da un amministratore. In pratica: chiedi «cerca le ultime notizie su X» e Claude porta risultati con i link, così risali alla fonte. Senza, risponde con ciò che «sa» e su fatti recenti può sbagliare.

Code execution e creazione file. Dà a Claude un ambiente sandbox per eseguire codice e produrre file veri: Excel, PowerPoint, Word, PDF, grafici. Serve anche per caricare file XLSX. È disponibile su tutti i piani, con un limite indicativo di ~30 MB per file. È ciò che permette richieste come «crea un Excel con queste spese e un grafico a torta»: Claude scrive il file e te lo restituisce pronto da scaricare.

Memory e ricerca nelle chat. La **Memory** crea una sintesi del tuo storico, così Claude costruisce sul contesto passato; è disponibile su tutti i piani. La **ricerca nelle chat passate** è invece riservata ai piani a pagamento. Le conversazioni in incognito (icona fantasma) restano escluse da entrambe.

Privacy e controllo dei dati

Tre cose da tenere a mente. Le chat in **incognito** non finiscono nello storico né nella Memory: usale quando non vuoi lasciare traccia. La **Memory** è una sintesi, non una copia fedele: se un dato è importante, ripetilo nella chat invece di confidare che sia stato ricordato. Infine, su **Team** ed **Enterprise** alcune capacità le abilita un amministratore da *Admin → Capabilities*: se non le vedi, non è un errore tuo, chiedi a chi gestisce l’organizzazione.

> **Tip:** non devi attivare tutto subito. Parti da Web search e Code execution: coprono la gran parte dei casi quotidiani; il resto lo aggiungi quando serve.

Projects, in due parole

Un **Project** è uno spazio di lavoro con una sua knowledge base e istruzioni dedicate, condivise tra le chat di quel progetto. È utile quando lavori a lungo sullo stesso tema. Gli utenti Free possono crearne fino a cinque; la condivisione è riservata a Team ed Enterprise. Li approfondiamo nel cap. L3.2.

Preferenze, Styles o Projects?

Si sovrappongono, ma servono a scopi diversi. Le **Instructions for Claude** sono preferenze personali valide ovunque. Gli **Styles** cambiano il modo di comunicare e li attivi all'occorrenza. Un **Project** isola contesto e istruzioni per un tema o un cliente specifico, senza inquinare le altre chat. Regola pratica: preferenze stabili → Instructions; come scrivere → Styles; un lavoro a sé → Project.

Esempio concreto: per scrivere questo libro potresti tenere un Project con le regole di stile e i capitoli già pronti, così ogni nuova chat parte già «sul pezzo» senza ripetere il contesto.

> **Nota:** le impostazioni viste qui sono personali. In un'organizzazione, alcune scelte (sicurezza, uso dei dati per il training) le definisce l'amministratore: il tuo account eredita quelle regole.

In pratica: configura il tuo Claude

1. Apri **Settings** → **Instructions for Claude** e scrivi nome, lingua e tono.
2. Scegli uno **stile** dal menu "Search and tools" (prova Concise).
3. Attiva **Web search** e fai una domanda su un fatto recente.
4. In **Settings** → **Capabilities**, verifica che **Code execution** sia attiva.

5. Controlla le impostazioni di **Memory** e decidi se tenerla attiva.
6. Se sei in un'organizzazione, verifica in **Admin → Capabilities** cosa è abilitato sul tuo account.

Errori comuni

- **Ripetere le stesse istruzioni a ogni chat.** Mettile una volta in "Instructions for Claude".
- **Dimenticare Web search sui fatti recenti.** Senza, Claude risponde "a memoria" e può sbagliare date o numeri.
- **Aspettarsi un file Excel senza Code execution.** Va attivata, altrimenti niente generazione di file.
- **Confidare nella Memory per tutto.** È una sintesi, non un archivio preciso: per dati importanti, riportali nella chat.
- **Pensare a un bug quando manca una funzione.** Su Team/Enterprise può averla disattivata l'amministratore: controlla in Admin → Capabilities.
- **Creare un Project per ogni domanda.** I Project servono ai lavori continui; per una richiesta isolata basta una chat normale.

Riepilogo

1. **Instructions for Claude** applica le tue preferenze a ogni conversazione.
2. Gli **Styles** regolano il tono; stanno evolvendo verso le Skills.
3. **Web search** serve per i dati aggiornati e consuma usage.
4. **Code execution** abilita la creazione di file; **Memory** ricorda lo storico.
5. La **ricerca nelle chat** è solo a pagamento; l'incognito resta escluso.

Prossimo passo

Hai completato i fondamenti. Nel **Livello 2** passiamo all’installazione locale: si parte dal **cap. L2.1 — Installare Claude Desktop**.

Capitolo L2.1 — Installare Claude Desktop

Livello 2 — Installazione locale. Dati di prodotto verificati il 22/06/2026 su fonti ufficiali.

Obiettivo

Al termine avrai l’app Claude Desktop installata sul tuo computer, saprai accedere e capirai a cosa servono i suoi tre tab: Chat, Cowork e Code. Claude Desktop è l’applicazione che porta Claude direttamente sul desktop, senza passare dal browser.

Prerequisiti

- Un account Claude (vedi cap. F.3). Per la sola **Chat** basta accedere; per **Cowork** serve un piano a pagamento.
- Un computer **macOS o Windows**. Su Linux l’app non esiste: si usa la CLI (vedi cap. L2.2).
- Connessione internet attiva.

Requisiti di sistema

L’app gira su sistemi recenti. Su Mac un’unica build copre sia i processori Intel sia Apple Silicon; su Windows c’è una versione a parte per ARM64.

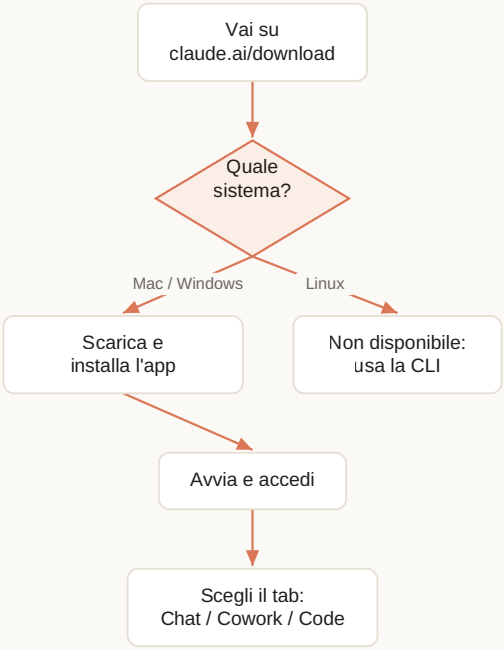
Tabella L2.1.1 — Sistemi supportati.

Sistema	Versione minima	Note
macOS	11 (Big Sur)	build Universal
Windows	10	x64; ARM64 a parte
Linux	—	non c'è: usa la CLI

Scaricare e installare

Il percorso è lo stesso per tutti: una pagina di download, un file da aprire, un accesso. La figura L2.1.1 riassume il flusso, compreso il caso Linux.

Figura L2.1.1 — Flusso di installazione di Claude Desktop. Alt-text: diagramma verticale dal download alla scelta del tab.



> **Nota:** la pagina di download riconosce il tuo sistema, ma puoi sempre scegliere a mano la versione macOS o Windows.

I tre tab dell'app

Una volta dentro, l'app si divide in tre aree. È utile sapere subito quale serve a cosa, così non cerchi una funzione nel posto sbagliato.

Tabella L2.1.2 — A cosa serve ogni tab.

Tab	A cosa serve
Chat	conversazioni con Claude
Cowork	task agentici lunghi e Dispatch
Code	sviluppo software in app

Chat è il punto di partenza. Cowork affida a Claude lavori in più passi su una cartella del tuo computer (lo vediamo nel Livello 3); da qui parte anche il **Dispatch** (l'invio di un task perché prosegua a distanza, ripreso al Livello 6). Code porta in app le capacità della CLI (Livello 2 e seguenti).

In pratica: installa e accedi

1. Apri **claude.ai/download** e scegli macOS o Windows.
2. **Apri il file** scaricato per completare l'installazione.
3. Avvia Claude da **Applicazioni** (Mac) o dal **menu Start** (Windows).
4. **Accedi** con il tuo account.
5. Apri i tab e prova la Chat con un primo messaggio.

> **Tip (Windows + Code):** la prima volta che apri il tab **Code** su Windows serve **Git for Windows** installato. Installalo e **riavvia** l'app.

Cowork: cosa sapere prima

Cowork è incluso nei piani a pagamento (Pro, Max, Team, Enterprise) e gira in una **VM isolata** (macchina virtuale) sul tuo computer. Legge e scrive solo nelle cartelle che colleghi, e la rete segue le tue impostazioni di egress (traffico in uscita). Su Windows va abilitata la **Virtual Machine Platform**.

Errori comuni

- **Sono su Linux e non trovo l'app.** Corretto: non esiste. Usa la CLI (cap. L2.2).
- **PC Windows ARM.** Scarica l'**installer ARM64** dedicato, non la x64.
- **Cowork non parte (Windows).** Verifica il piano a pagamento e che la **Virtual Machine Platform** sia abilitata.
- **Il tab Code dà errore al primo avvio (Windows).** Manca Git for Windows: installalo e riavvia l'app.

Riepilogo

1. Claude Desktop è disponibile su **macOS 11+** e **Windows 10+**, non su Linux.
2. Si installa da **claude.ai/download**: scarica, apri, accedi.
3. L'app ha tre tab: **Chat**, **Cowork**, **Code**.
4. **Cowork** richiede un piano a pagamento e gira in una VM isolata.
5. Su Windows, il tab **Code** richiede **Git for Windows** alla prima apertura.

Prossimo passo

Nel **cap. L2.2 — Installare Claude Code** vediamo la versione da riga di comando (CLI), utile su Linux e per chi automatizza, e i metodi di installazione con la relativa manutenzione.

Dati verificati il 22/06/2026 su support.claude.com (Install Claude Desktop) e code.claude.com/docs/en/desktop. L'installazione è grafica e legata all'account, quindi non è stata eseguita nella VM; i passaggi sono riportati fedelmente dalla documentazione ufficiale.

Capitolo L2.2 — Installare Claude Code

Livello 2 — Installazione locale. Dati di prodotto verificati il 21/06/2026 su fonti ufficiali.

Obiettivo

Al termine avrai Claude Code installato sul tuo computer, saprai scegliere il metodo giusto per il tuo sistema operativo e avrai verificato che funzioni. Claude Code e lo strumento da riga di comando (CLI) che lavora direttamente sui file del tuo progetto.

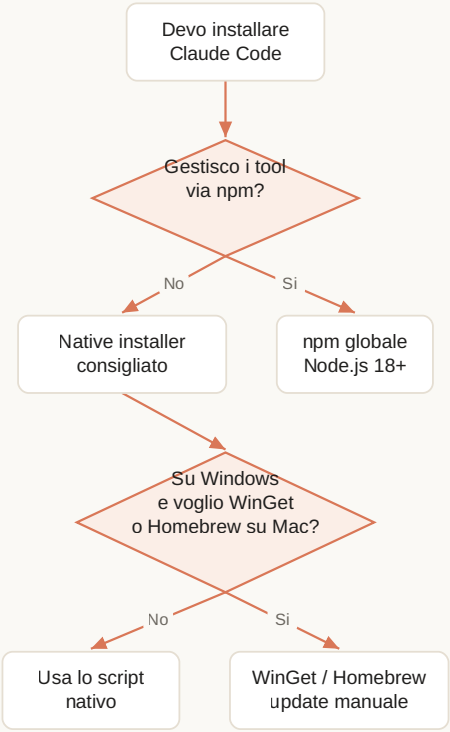
Prerequisiti

- Un account a pagamento: **Pro, Max, Team, Enterprise o Console (API)**. Il piano Free non include Claude Code (vedi cap. F.3 e il ledger).
- Un terminale aperto. Se non l'hai mai usato, vedi il cap. L2.1.
- Connessione internet attiva.

Quale metodo scegliere

Esistono piu modi di installare, ma per quasi tutti la scelta e semplice: il **native installer**. Non richiede Node.js e si aggiorna da solo in background. Gli altri metodi servono in casi specifici.

Figura L2.2.1 — Albero di decisione per il metodo di installazione. Alt-text: diagramma verticale che guida dalla scelta del metodo.



> **Nota:** native installer, Homebrew, WinGet e npm installano **lo stesso binario**. Cambia solo come arriva e come si aggiorna.

Installazione con il native installer (consigliato)

Apri il terminale ed esegui il comando per il tuo sistema.

macOS, Linux, WSL

```
curl -fsSL https://claude.ai/install.sh | bash
```

Windows — PowerShell

```
irm https://claude.ai/install.ps1 | iex
```

Windows — CMD

Il comando va su **una riga sola**; qui e spezzato con `^` per la stampa.

```
curl -fsSL https://claude.ai/install.cmd ^
  -o install.cmd && install.cmd ^
  && del install.cmd
```

> **Attenzione (Windows):** se vedi l'errore *"The token '&&' is not a valid statement separator"* sei in PowerShell, non in CMD. Il prompt mostra `PS C:\` in PowerShell e `C:\` senza `PS` in CMD.

Su Windows nativo, **Git for Windows** e opzionale ma consigliato: abilita il Bash tool. Senza Git, Claude Code usa il PowerShell tool.

Metodi alternativi

Tabella L2.2.1 — Quando usare un metodo diverso.

Metodo	Comando	Aggiornamento
Homebrew (mac)	`brew install --cask claude-code`	manuale
WinGet (Win)	`winget install Anthropic.ClaudeCode`	manuale
npm	`npm install -g @anthropic-ai/claude-code`	manuale

Per npm serve **Node.js 18+**. Regola d'oro: **mai** `sudo npm install -g`, perche crea problemi di permessi e rischi di sicurezza. Per aggiornare usa `npm install -g @anthropic-ai/claude-code@latest`, non `npm update -g`.

Su Debian/Ubuntu, Fedora/RHEL e Alpine esistono anche repository firmati (apt, dnf, apk) su `downloads.claude.ai`: utili in azienda, si aggiornano con il normale gestore di sistema.

In pratica: installa e verifica

1. Esegui il comando del native installer per il tuo sistema.
2. **Apri un nuovo terminale** (per ricaricare il PATH).
3. Controlla la versione:

   ```
   claude --version
   ```

4. Esegui la diagnosi di installazione e configurazione:

   ```
   claude doctor
   ```

5. Entra in una cartella di **progetto reale** (non vuota) e avvia:

   ```
   claude
   ```

   Al primo avvio si apre il browser per il login: segui le istruzioni.

Errori comuni

- `command not found` **dopo l'installazione.** Apri un nuovo terminale. Il binario nativo sta in `~/.local/bin/claude`: assicurati che quella cartella sia nel PATH.
- **Tab/login rifiutato.** Verifica che l'account sia a pagamento: il Free non include Claude Code.
- **Errori di permessi con npm.** Non usare `sudo`. Punta il prefix di npm a una cartella tua o usa nvm.

- **Git Bash non trovato (Windows).** Imposta il percorso in `settings.json`:

```
{
  "env": {
    "CLAUDE_CODE_GIT_BASH_PATH":
      "C:\\Program Files\\Git\\bin\\bash.exe"
  }
}
```

Riepilogo

1. Per quasi tutti, il **native installer** e la scelta giusta: niente Node, auto-update.
2. Tre comandi diversi per macOS/Linux/WSL, PowerShell e CMD.
3. npm, Homebrew e WinGet installano lo stesso binario, ma si aggiornano a mano.
4. Serve un account a pagamento; **mai** `sudo` con npm.
5. Verifica sempre con `claude --version` e `claude doctor`.

Prossimo passo

Nel **cap. L2.3 — Autenticazione e controlli** completiamo il login, vediamo l'uso con API key negli ambienti automatizzati e leggiamo l'output di `claude doctor` per risolvere i problemi piu frequenti.

Comandi verificati su code.claude.com/docs/en/setup il 21/06/2026. Eseguibilita nella VM: gli script di installazione richiedono rete verso claude.ai e un account, quindi non sono stati eseguiti qui; i comandi sono riportati fedelmente dalla documentazione ufficiale.

Capitolo L2.3 — Autenticazione e controlli

Livello 2 — Installazione locale. Dati di prodotto verificati il 22/06/2026 su fonti ufficiali.

Obiettivo

Al termine saprai accedere a Claude Code con il metodo giusto — abbonamento o API key — capire quale credenziale è attiva e diagnosticare i problemi più comuni di login e di PATH. È il passo che trasforma un'installazione in uno strumento pronto all'uso.

Prerequisiti

- Claude Code installato e funzionante (vedi cap. L2.2).
- Un account a pagamento o una API key da Console (vedi cap. F.3).

Il primo accesso

Dopo l'installazione, entra in una cartella di progetto ed esegui:

```
claude
```

Al primo avvio Claude Code apre il browser per il login (OAuth, accesso delegato senza digitare la password nel terminale). Accedi con il tuo account Claude.ai e torni al terminale già autenticato. Se il browser non si apre da solo, premi `c` per copiare l'URL e incollarlo a mano.

A seconda di chi sei, accedi con un abbonamento **Pro o Max** (account Claude.ai), un account **Team/Enterprise** invitato dall'amministratore, oppure credenziali **Claude Console** se la tua organizzazione fattura via API.

> **Nota:** per uscire e ri-autenticarti digita `/logout` al prompt di Claude Code; `/login` riavvia l'accesso.

Due strade: abbonamento o API key

Le due modalità più comuni rispondono a bisogni diversi. L'**abbonamento** (login da browser) è la scelta normale quando lavori al computer. La **API key** serve quando un browser non c'è: server, pipeline di CI, script automatici. Lì il login interattivo non è praticabile, e si usa una chiave generata nella Console.

Figura L2.3.1 — Come scegliere il metodo di accesso. Alt-text: diagramma verticale che dal comando claude porta a tre vie di accesso e poi alla verifica con lo stato.

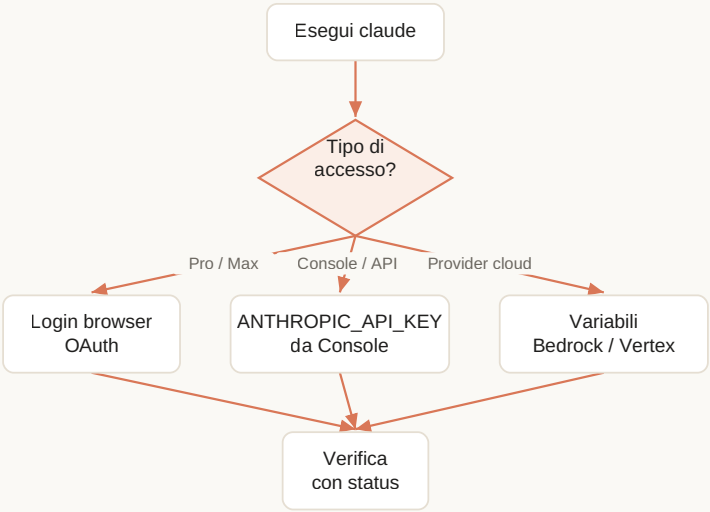


Tabella L2.3.1 — I metodi di autenticazione.

Metodo	Per chi	Come
OAuth	uso al computer	login browser
API key	headless / CI	ANTHROPIC_API_KEY
Cloud	Bedrock/Vertex	variabili dedicate

Usare una API key headless

Genera la chiave nella Console (platform.claude.com) e impostala come variabile d'ambiente. Su macOS e Linux:

```
export ANTHROPIC_API_KEY=sk-ant-...
```

Su Windows (PowerShell):

```
$env:ANTHROPIC_API_KEY = "sk-ant-..."
```

In modalità interattiva, Claude Code ti chiede una volta di approvare la chiave e ricorda la scelta. In modalità non interattiva (`claude -p`) la chiave viene usata sempre, senza chiedere: è ciò che serve negli script.

Quale credenziale vince

Se più credenziali sono presenti, Claude Code ne sceglie una con un ordine fisso, dalla più forte alla più debole:

1. provider cloud (`CLAUDE_CODE_USE_BEDROCK` e simili);
2. `ANTHROPIC_AUTH_TOKEN` (bearer per gateway o proxy);
3. `ANTHROPIC_API_KEY`;
4. script `apiKeyHelper` (credenziali a rotazione);
5. abbonamento OAuth da `/login`.

Conoscere l'ordine spiega la trappola più frequente: se hai un abbonamento attivo **ma** anche `ANTHROPIC_API_KEY` impostata, vince la

key. Se quella chiave appartiene a un'organizzazione disabilitata, il login fallisce pur avendo un abbonamento valido.

> **Attenzione:** queste variabili valgono solo per la CLI da terminale. Claude Desktop e le sessioni remote usano sempre OAuth e ignorano `ANTHROPIC_API_KEY`.

Dove finiscono le credenziali

Claude Code conserva il login in modo sicuro: su macOS nel **Keychain** cifrato; su Linux e Windows nel file `~/.claude/.credentials.json` (su Linux con permessi `0600`), oppure sotto `$CLAUDE_CONFIG_DIR` se impostata. Non devi gestirlo a mano, ma sapere dov'è aiuta per backup o reset.

In pratica: accedi e verifica

1. Entra in una cartella di progetto ed esegui `claude`.
2. Completa il login nel browser (o premi `c` per copiare l'URL).
3. Controlla quale metodo è attivo con `/status`.
4. Se serve la diagnosi dell'installazione:

   ```
   claude doctor
   ```

5. In un ambiente automatizzato imposta `ANTHROPIC_API_KEY` e avvia con `claude -p`.

Errori comuni

- **Login fallito con abbonamento valido.** Probabile `ANTHROPIC_API_KEY` di troppo: esegui `unset ANTHROPIC_API_KEY` e ricontrolla con `/status`.

- `command not found` **dopo l'installazione.** Apri un nuovo terminale; il binario è in `~/.local/bin/claude`, che deve stare nel PATH (vedi cap. L2.2).
- **Il browser non si apre al login.** Premi `c` per copiare l'URL e aprilo a mano.
- **API key ignorata nel Desktop.** È normale: il Desktop usa solo OAuth.

Riepilogo

1. Al primo `claude` si accede dal **browser** con l'account Claude.ai.
2. Per gli ambienti **headless** si usa `ANTHROPIC_API_KEY` dalla Console.
3. Con più credenziali vince un **ordine fisso**: la API key batte l'abbonamento.
4. `/status` dice quale metodo è attivo; `/logout` ripulisce l'accesso.
5. `claude doctor` diagnostica installazione e configurazione.

Prossimo passo

Nel **cap. L2.4 — Configurare il progetto** prepariamo `CLAUDE.md`, la cartella `.claude` e i permessi, per far partire Claude Code già "sul pezzo" su un repo reale.

Dati verificati il 22/06/2026 su code.claude.com/docs/en/authentication e support.claude.com (troubleshoot Claude Code). Login OAuth e API key richiedono un account reale, quindi non sono stati eseguiti nella VM; comandi e variabili sono riportati fedelmente dalla documentazione ufficiale.

Capitolo L2.4 — Configurare il progetto

Livello 2 — Installazione locale. Concetti stabili; dettagli di prodotto verificati su fonti ufficiali.

Obiettivo

Al termine saprai preparare una cartella di progetto perche Claude Code ci lavori bene: scrivere un file `CLAUDE.md` che dà il contesto, capire a cosa serve la cartella `.claude/`, e impostare i permessi così Claude agisce senza chiederti conferma a ogni passo, ma neanche di nascosto.

Prerequisiti

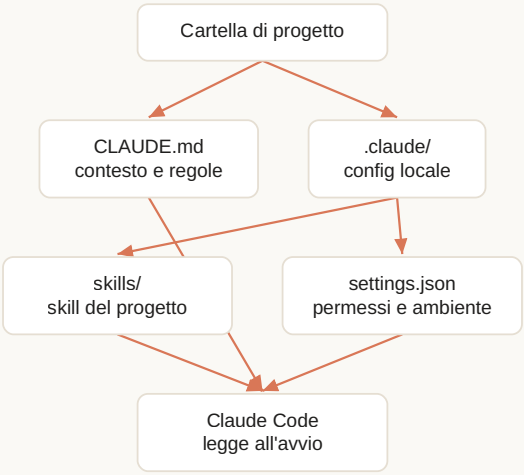
- Claude Code installato e funzionante (vedi cap. L2.2).
- Login completato (vedi cap. L2.3).
- Una cartella di progetto **reale**: codice, documenti, qualunque cosa su cui vuoi lavorare. Claude Code rende meglio su un progetto vero che su una cartella vuota.

Perche serve configurare

Claude Code parte senza sapere nulla del tuo progetto. Ogni volta che apri una sessione, ricostruisce il contesto leggendo i file. Puoi spiegargli tutto a voce a ogni avvio, oppure scriverlo una volta in un file che legge da solo. La seconda strada è quella che ripaga: meno ripetizioni, comportamento più prevedibile, e le stesse regole valgono per chiunque apra il progetto.

Tre elementi fanno la configurazione: il file `CLAUDE.md` (il contesto), la cartella `.claude/` (skill, comandi, impostazioni locali) e i permessi (cosa Claude può fare senza chiedere).

Figura L2.4.1 — Cosa legge Claude Code all'avvio in una cartella di progetto. Alt-text: diagramma verticale che mostra la cartella di progetto con CLAUDE.md e la sottocartella .claude e il loro ruolo.



Il file CLAUDE.md

`CLAUDE.md` è un file di testo in formato Markdown che metti nella radice del progetto. Claude Code lo legge a ogni avvio e ne tratta il contenuto come istruzioni da seguire. È il posto giusto per le cose che non vuoi ripetere:

- **Cos'è il progetto:** una frase o due su scopo e struttura.
- **Comandi che usi spesso:** come si avvia, come si testa, come si compila.
- **Convenzioni:** stile del codice, nomi, cose da non toccare.

- **Vincoli:** “non modificare la cartella X”, “i commenti in inglese”.

La regola è: poco e utile. Un `CLAUDE.md` enorme si diluisce; uno mirato pesa su ogni risposta. Scrivi le istruzioni come le diresti a una persona nuova nel team al primo giorno.

> **Tip:** dentro Claude Code, il comando `/init` analizza il progetto e ti propone un primo `CLAUDE.md`. È un buon punto di partenza da rifinire a mano, non un risultato finito.

Un esempio minimo:

```
# Progetto: API ordini

App Node.js. Avvio: `npm run dev`.
Test: `npm test`. Non modificare `db/migrations/`.
Commenti nel codice in inglese.
```

La cartella .claude/

Accanto a `CLAUDE.md` puoi avere una cartella `.claude/`. Raccoglie la configurazione locale del progetto:

- `.claude/skills/` — le skill specifiche di questo progetto (vedi Livello 5).
- `.claude/settings.json` — permessi e variabili d’ambiente del progetto.

Il vantaggio è che questa configurazione vive **dentro** il progetto: se è in un repository git, la condividi con il team e la versioni come il resto. Chi clona il progetto eredita le stesse regole.

Permessi: il giusto mezzo

Claude Code chiede conferma prima di azioni che cambiano qualcosa: modificare un file, eseguire un comando. È una protezione, ma confermare tutto rallenta. I permessi servono a dire in anticipo “queste cose le puoi fare senza chiedere, queste mai”.

Si impostano in `.claude/settings.json`. Lo schema di base distingue ciò che è sempre permesso (`allow`) da ciò che è sempre negato (`deny`):

```
{
  "permissions": {
    "allow": ["Bash(npm test)"],
    "deny": ["Bash(rm -rf *)"]
  }
}
```

La filosofia: concedi le azioni ripetitive e sicure (lanciare i test, leggere file), tieni la conferma per quelle che spostano o cancellano. Non aprire tutto “per comodità”: il senso del passaggio di conferma è che tu resti l’ultima parola sulle azioni che contano.

In pratica: prepara una cartella

1. Apri il terminale ed entra nel tuo progetto:

   ```
   cd ~/progetti/mia-app
   ```

2. Avvia Claude Code:

   ```
   claude
   ```

3. Genera una bozza di contesto:

   ```
   /init
   ```

4. Apri `CLAUDE.md`, taglia il superfluo, aggiungi i comandi che usi e le cose da non toccare.
5. Se hai regole su comandi ricorrenti, crea `.claude/settings.json` con i permessi `allow` / `deny`.
6. Se il progetto è in git, fai un commit: ora la configurazione viaggia con il codice.

Errori comuni

- **CLAUDE.md troppo lungo.** Diventa rumore. Tieni solo ciò che cambia il comportamento: scopo, comandi, vincoli.
- **Permessi troppo larghi.** Mettere tutto in `allow` annulla la rete di sicurezza. Concedi il ripetitivo e sicuro, non il distruttivo.
- **Configurazione fuori dal repo.** Se `.claude/` e `CLAUDE.md` non sono versionati, il team non li eredita e le regole valgono solo per te.
- **Partire da una cartella vuota.** Claude Code dà il meglio su un progetto reale. Per imparare, usa un repo che già conosci.

Riepilogo

1. `CLAUDE.md` è il contesto che Claude Code legge a ogni avvio: scopo, comandi, convenzioni, vincoli.
2. Tienilo corto e mirato: poco e utile batte lungo e generico.
3. La cartella `.claude/` raccoglie skill e impostazioni locali del progetto.
4. I permessi (`allow` / `deny`) bilanciano velocità e controllo: concedi il sicuro, conferma il distruttivo.
5. Versiona `CLAUDE.md` e `.claude/` nel repo: così le regole valgono per tutti.

Prossimo passo

Con l’installazione locale completa, il **Livello 3 — Lavoro quotidiano** entra nell’uso di tutti i giorni. Si parte dal **cap. L3.1 — Cowork: primi passi**, dove deleghi a Claude un compito su una cartella e impari ad approvare le sue azioni.

Concetti (CLAUDE.md, permessi, struttura del progetto) verificati su code.claude.com/docs il 24/06/2026. I comandi `/init` e l’avvio di `claude` richiedono un account attivo e non sono stati eseguiti in questa sede.

Capitolo L3.1 — Cowork: primi passi

Livello 3 — Lavoro quotidiano. Dati di prodotto verificati il 27/06/2026 su fonti ufficiali.

Obiettivo

Al termine saprai cos'è Cowork e in cosa differisce da Chat e da Claude Code, avviare un task su una cartella del tuo computer, descrivere un risultato invece di una sequenza di passi, e approvare le azioni che Claude propone mentre lavora.

Prerequisiti

- Claude Desktop installato (vedi cap. L2.1).
- Un piano a pagamento: **Pro, Max, Team o Enterprise**. Cowork non è incluso nel Free (vedi cap. F.3 e il ledger).
- Una cartella con file su cui ha senso lavorare.

Cos'è Cowork

Cowork è il tab dell'app desktop per il **lavoro agentico non di codice**: gli affidi un compito su una cartella e lui lo porta avanti da solo — legge i file, ragiona, crea o modifica documenti, esegue passaggi in sequenza — fermandosi a chiederti conferma quando serve. È disponibile sui piani a pagamento su macOS e Windows. (VOLATILE: lo stato del prodotto evolve in fretta; vedi il ledger.)

Gira in una **VM isolata** (una macchina virtuale, un computer dentro il computer) sul tuo dispositivo. Questo è il punto chiave per la fiducia: Cowork legge e scrive **solo nelle cartelle che gli colleghi**, e la rete se-

gue le impostazioni di egress (cosa può raggiungere su internet). Non vede il resto del tuo disco se non glielo dai.

Cowork, Chat e Code: chi fa cosa

I tre tab dell'app rispondono a bisogni diversi. Sceglierli bene è metà del lavoro.

Tabella L3.1.1 — Quando usare quale tab.

Tab	Per cosa	Forma
Chat	Domande, bozze, idee	Botta e risposta
Code	Sviluppo software	Sui file di codice
Cowork	Compiti su file, multi-passo	Agentico, lungo

In breve: se vuoi una risposta, Chat. Se programmi, Code. Se vuoi che qualcuno **faccia** una cosa articolata sui tuoi file — riorganizza una cartella, produca un report da più documenti, prepari una serie di file — Cowork.

End-state, non i passi

L'errore più comune al primo uso è guidare Cowork passo per passo, come si fa in chat. Funziona meglio il contrario: descrivi il **risultato che vuoi** (lo "end-state") e lascia che sia lui a trovare la sequenza.

- **Debole (passi):** "Apri il file. Ora leggi la colonna B. Adesso copia…"
- **Meglio (end-state):** "Da questi tre Excel di vendita, crea un riepilogo per regione con i totali trimestrali, salvato come `riepilogo.xlsx`."

Descrivere il risultato sfrutta ciò per cui Cowork è fatto: pianificare il percorso. Tu definisci il "dove arrivare" e i vincoli; lui trova il "come".

Approvare le azioni

Mentre lavora, Cowork si ferma prima delle azioni che contano e ti mostra cosa sta per fare: creare un file, modificarne uno, eseguire un comando. Tu approvi o correggi. È lo stesso principio dei permessi di Claude Code (vedi cap. L2.4): la parte ripetitiva scorre, le decisioni restano tue.

Non approvare a scatola chiusa. Leggi cosa propone, soprattutto le prime volte: è così che impari a fidarti — o a capire dove serve essere più precisi nella richiesta.

Cowork ha due **modalità di permessi** che decidono quanto spesso si ferma:

- **Ask before acting:** si ferma e chiede conferma a ogni azione. È la modalità giusta con file o strumenti nuovi, o quando vuoi tenere tutto sotto controllo.
- **Act without asking:** procede senza fermarsi. Più veloce, ma più rischiosa: usala solo mentre stai supervisionando, su file e siti di cui ti fidi.

In **entrambe** le modalità, Claude chiede sempre conferma prima di **cancellare** file in modo definitivo: la cancellazione non è mai automatica.

In pratica: il tuo primo task

1. Apri Claude Desktop e vai sul tab **Cowork**.
2. **Collega una cartella**: scegline una con dei file veri, ma di cui hai una copia. Cowork agirà solo dentro questa cartella.
3. Scrivi il compito come **end-state**, con i vincoli:

```
In questa cartella ci sono appunti .txt sparsi.
Raggruppali per argomento in sottocartelle e
crea un INDICE.md che elenca cosa c'è dove.
```

4. Lascia che pianifichi. Quando si ferma per un'azione, **leggi e approva**.
5. A fine task, controlla il risultato nella cartella. Se non è quello che volevi, raffina la richiesta e rilancia.

Errori comuni

- **Guidarlo passo per passo.** Spreca il suo punto di forza. Descrivi il risultato, non la procedura.
- **Collegare l'intero disco o cartelle critiche.** Collega solo ciò che serve, e lavora su copie finché non ti fidi.
- **Aspettarsi Cowork nel Free.** Serve un piano a pagamento.
- **Approvare senza leggere.** Le prime volte controlla cosa propone: è lì che impari a dargli istruzioni migliori.

Riepilogo

1. Cowork è il tab desktop per il lavoro agentico non di codice, in una **VM isolata** che vede solo le cartelle collegate.
2. Chat per le risposte, Code per il software, Cowork per i compiti multi-passo sui tuoi file.
3. Descrivi il **risultato** (end-state) e i vincoli, non la sequenza di passi.
4. Approva le azioni che Cowork propone: il controllo resta tuo.
5. Collega solo le cartelle necessarie e parti da copie.

Prossimo passo

Nel **cap. L3.2 — Projects** vediamo come dare a un lavoro ricorrente una casa stabile: istruzioni, file di riferimento e memoria che restano tra una sessione e l’altra.

Dati su Cowork (VM isolata, piani, cartelle collegate, modalità di permessi) dal ledger, verificati il 27/06/2026 su support.claude.com/en/articles/13345190 e code.claude.com/docs. Il task di esempio non è stato eseguito qui: richiede l’app desktop con un account a pagamento.

Capitolo L3.2 — Projects

Livello 3 — Lavoro quotidiano. Dati di prodotto verificati il 22/06/2026 su fonti ufficiali.

Obiettivo

Al termine saprai cos’è un Project, quando conviene crearne uno, come dargli istruzioni e file di riferimento che restano tra una conversazione e l’altra, e cosa cambia quando lavori in team. Useremo come esempio il Project con cui è stato scritto questo libro.

Prerequisiti

- Saper conversare in chat (vedi cap. L1.2).
- Un’attività ricorrente: qualcosa su cui torni più volte, non una domanda secca.

Cos’è un Project

Un Project è un **workspace** dedicato a un’attività: raccoglie in un posto solo le istruzioni che valgono sempre, i file di riferimento (la “knowledge base”) e le conversazioni collegate. Ogni chat aperta dentro il Project parte già con quel contesto, senza che tu lo ripeta.

Li avevi già incontrati di sfuggita tra le impostazioni (cap. L1.3): qui li mettiamo al lavoro. I Project sono disponibili su tutti i piani; nel **Free** sono limitati a **5**. La condivisione in team è riservata a **Team ed Enterprise** (vedi il ledger).

Quando crearne uno

Non tutto merita un Project. La singola domanda sta bene in una chat normale. Conviene creare un Project quando ricorrono questi segnali:

- **Torni più volte** sullo stesso lavoro nel tempo.
- **Ripeti lo stesso contesto** a ogni chat (“ricordati che lo stile è…”).
- **Hai file di riferimento** che servono in ogni conversazione.

Se ti ritrovi a incollare le stesse istruzioni o gli stessi documenti all’inizio di ogni chat, è il momento del Project.

Istruzioni e knowledge base

Due cose rendono utile un Project:

- **Istruzioni:** valgono per ogni chat del Project. Tono, vincoli, formato, cose da fare e da non fare. È il “chi sei e come lavori” del progetto.
- **Knowledge base:** i file che Claude può consultare — documenti, esempi, materiale di riferimento. Diventano parte del contesto disponibile.

La differenza con `CLAUDE.md` (cap. L2.4) è il contesto d’uso: `CLAUDE.md` governa Claude Code sui file del progetto; le istruzioni del Project governano le **conversazioni** dentro quel Project. Stessa idea — scrivi una volta il contesto stabile — applicata a strumenti diversi.

> **Nota:** i file della knowledge base hanno un limite per file (intorno ai 30 MB) ma numero non vincolato, purché il contenuto stia nel contesto di Claude. È un limite diverso da quello degli allegati in chat (cap. L1.1): la knowledge base è pensata per molti file di riferimento, non per pochi file enormi.

Memoria e scope

Il Project dà ai tuoi lavori un confine. Le istruzioni e i file restano **circoscritti** a quel Project: non si mescolano con le altre chat. Questo è un vantaggio doppio — il contesto giusto è sempre lì, e quello sbagliato resta fuori. Quando cambi attività, cambi Project, e il modello non trascina dietro materiale che non c'entra.

Esempio: il Project di questo libro

Questo manuale è stato scritto dentro un Project. La sua forma mostra bene il pattern:

- **Istruzioni:** regole di progetto — dove salvare i capitoli, niente note prudenziali nei testi pubblicati, voce e vincoli di impaginazione.
- **Knowledge base:** il piano editoriale, il ledger dei fatti, i file di contesto su voce e pubblico.
- **Conversazioni:** una per blocco di capitoli, tutte già allineate alle stesse regole senza ripeterle.

Risultato: ogni sessione di scrittura parte sapendo com'è fatto il libro. È la differenza tra spiegare il progetto da capo ogni volta e averlo scritto una volta sola.

In pratica: crea il tuo Project

1. In Claude, apri la sezione **Projects** e creane uno nuovo.
2. Dagli un nome che dice di cosa si tratta.
3. Scrivi le **istruzioni**: il contesto stabile, i vincoli, il formato che vuoi.
4. Carica nella **knowledge base** i file di riferimento ricorrenti.
5. Apri una chat dentro il Project e lavora: il contesto è già attivo.
6. Quando una regola si ripete in più chat, spostala nelle istruzioni del Project.

Errori comuni

- **Un Project per ogni chat.** Sono per il lavoro ricorrente, non per la domanda singola. Troppi Project diventano disordine.
- **Istruzioni-fiume.** Vale come per `CLAUDE.md`: poco e mirato. L'eccesso si diluisce.
- **Knowledge base obsoleta.** I file di riferimento invecchiano. Aggiornali, o Claude lavora su materiale vecchio.
- **Aspettarsi la condivisione nel Free/Pro.** Condividere un Project in team è da Team/Enterprise.

Riepilogo

1. Un Project è un workspace con istruzioni, file di riferimento e chat collegate, tutto in un posto.
2. Crealo quando torni più volte su un lavoro e ripeti lo stesso contesto.
3. Le istruzioni valgono per ogni chat del Project; la knowledge base è il materiale che Claude consulta.
4. Lo scope tiene dentro il contesto giusto e fuori quello che non c'entra.
5. Free fino a 5 Project; condivisione in team su Team/Enterprise.

Prossimo passo

Nel **cap. L3.3 — Connettori** colleghiamo Claude alle app che già usi — Drive, Slack, gestori di task — così può leggere i tuoi dati e agire al loro interno.

Dati sui Project (limiti Free, condivisione team) dal ledger, verificati il 22/06/2026 su support.claude.com. L'esempio del Project di questo libro è reale; i passaggi operativi non sono stati eseguiti in questa sede.

Capitolo L3.3 — Connettori

Livello 3 — Lavoro quotidiano. Dati di prodotto verificati il 24/06/2026 su fonti ufficiali.

Obiettivo

Al termine saprai cosa sono i Connectors (connettori), come collegarne uno dalla directory, come attivarli in una conversazione, e quali cautele tenere — sia da utente sia, in azienda, da amministratore.

Prerequisiti

- Saper aprire una chat (vedi cap. L1.1) e usare il menu strumenti (vedi cap. L1.3).
- Un account su un servizio che vuoi collegare (Drive, Slack, un gestore di task…).

Cosa sono i connettori

Un connettore dà a Claude accesso a un’app o un servizio: leggere i tuoi dati e **compiere azioni** al loro interno. Collega Claude a Linear per aprire issue, a Slack per inviare messaggi, a Google Drive per cercare nei tuoi file.

Il punto da capire bene è il modello dei permessi: **Claude eredita i tuoi permessi** nel servizio collegato. Se tu non puoi vedere un certo file, canale o record alla fonte, il connettore non lo raggiunge da Claude. Un connettore non ti dà più accesso di quanto già hai — lo porta solo dentro la conversazione.

I web connectors sono disponibili per **tutti gli utenti** su Claude, Cowork, Desktop e Mobile; funzionano anche con Claude Code e via API.

La directory dei connettori

I connettori disponibili stanno nella **Connectors Directory** (claude.ai/connectors). Ogni connettore ha una scheda con i casi d'uso, le capacità di lettura/scrittura e la disponibilità. Alcuni hanno il badge **Interactive**: rendono interfacce live — dashboard, board, strumenti — dentro la conversazione.

Apri la directory in due modi:

- **Da una chat:** pulsante **"+"** (o "/") > "Connectors" > "Manage connectors" > "+".
- **Dalle impostazioni: Customize > Connectors** > "+".

Collegare e attivare un servizio

Collegare un servizio e usarlo sono due passaggi distinti.

Collegare (una volta): dalla directory, clic sul connettore, rivedi le sue capacità, "Connect"/"Install", segui i prompt di autenticazione (di solito OAuth, cioè autorizzi Claude dal sito del servizio), configura i permessi.

Attivare (per conversazione): "+" (o "/") > "Connectors", poi accendi con il toggle i servizi che vuoi per quella chat. Da lì Claude li usa quando sono pertinenti — e può proporli da solo, senza che tu li nomini ogni volta.

> **Tip:** se hai molti connettori, da "Connectors" > "Tool access" scegli come si caricano. Il default **Auto** va bene per quasi tutti; con 10 o più connettori attivi, **On demand** lascia più spazio alla conversazione.

In azienda: il ruolo dell'admin

Su **Team ed Enterprise** un connettore non è usabile finché un **Owner** o **Primary Owner** non lo abilita per l'organizzazione (Organization settings > Connectors > "Browse connectors" > "Add to your team"). Abilitare **non** dà accesso a nessuno: ognuno si autentica comunque per conto suo.

L'admin può anche **restringere le azioni** di un connettore per tutta l'org. Da Customize > Connectors, sul connettore, le **Tool permissions** sono divise per tipo (sola lettura, scrittura/cancellazione) e per ciascuna scegli *Always allow*, *Needs approval* o *Blocked*. Vale per tutti e non è aggirabile dal singolo.

Tabella L3.3.1 — Esempi di restrizione delle azioni.

Servizio	Permetti	Blocca
Email	Cercare e riassumere	Inviare messaggi
Drive	Leggere i file	Creare/modificare
Linear	Vedere le issue	Crearne o cambiarle

Queste restrizioni lavorano **insieme** ai permessi del servizio: anche dove Claude può scrivere, serve comunque il permesso alla fonte. Restringere in Claude non concede mai più di quanto il sistema d'origine consente — al massimo restringe.

Cautele

- **Connetti solo ciò di cui ti fidi e che ti serve.** Collegare un servizio significa dare a Claude accesso ai tuoi dati lì dentro.
- **Rivedi gli accessi richiesti** durante la connessione, e **disconnetti** ciò che non usi più (Customize > Connectors).

- **Custom connector = non verificato da Anthropic.** Puoi aggiungere connettori personalizzati (remote MCP), ma collega solo server di organizzazioni fidate. Sui custom connector entriamo nel dettaglio al Livello 6 (cap. L6.2).
- **Team/Enterprise:** i connettori funzionano solo in Project privati, e le chat con contenuti sincronizzati non sono condivisibili.

In pratica: collega Google Drive

1. In chat, “+” (o “/”) > “Connectors” > “Manage connectors” > “+”.
2. Cerca il connettore nella directory e aprilo.
3. Leggi le capacità, poi “Connect” e autorizza con il tuo account (OAuth).
4. Torna in chat, “+”, “Connectors”, e attiva il toggle del servizio.
5. Prova:

```
Cerca nel mio Drive il preventivo di marzo
e riassumimi le voci principali.
```

Errori comuni

- **“Ho collegato ma non lo usa.”** Collegare non basta: attiva il toggle nel menu “Connectors” della chat.
- **Aspettarsi accesso a tutto.** Claude vede solo ciò che vedi tu nel servizio.
- **(Team) “Il connettore non c’è.”** Deve prima abilitarlo un Owner per l’org; poi ti autentichi tu.
- **Custom connector che va in timeout.** Il server MCP è raggiunto dal cloud di Anthropic, non dal tuo device: deve stare su internet pubblico (vedi cap. L6.2).

Riepilogo

1. Un connettore dà a Claude accesso a un’app per leggere dati e agire, **con i tuoi stessi permessi** alla fonte.
2. I connettori stanno nella Connectors Directory; alcuni sono **Interactive**.
3. Collegare (OAuth, una volta) e attivare (toggle, per conversazione) sono due passi distinti.
4. In Team/Enterprise un Owner abilita il connettore e può restringerne le azioni per tutta l’org.
5. Connetti solo ciò di cui ti fidi, rivedi gli accessi, disconnetti l’inutile.

Prossimo passo

Nel **cap. L3.4 — Documenti** usiamo Claude per generare file professionali — Word, PowerPoint, Excel, PDF — partendo dal contenuto giusto.

Dati sui connettori (directory, OAuth, ruolo admin, restrizioni azioni) dal ledger, verificati il 24/06/2026 su support.claude.com/en/articles/11176164. I passaggi operativi non sono stati eseguiti in questa sede.

Capitolo L3.4 — Documenti (Word, PowerPoint, Excel, PDF)

Livello 3 — Lavoro quotidiano. Dati di prodotto verificati il 22/06/2026 su fonti ufficiali.

Obiettivo

Al termine saprai far generare a Claude documenti professionali — Word, Excel, PowerPoint, PDF — chiedendoli nel modo giusto, sfruttando il principio “prima il contenuto, poi il file”, e partendo da un modello quando vuoi un risultato ripetibile.

Prerequisiti

- Saper scrivere richieste chiare (vedi cap. L1.2).
- Avere attiva la **Code execution / file creation** (vedi cap. L1.3): è la sandbox con cui Claude crea i file. È su tutti i piani.

Come Claude crea i file

Claude non “scrive a mano” un .docx: usa una sandbox di esecuzione codice che genera il file vero, con la formattazione che serve. Per questo i documenti prodotti sono file reali — apribili in Word, Excel, PowerPoint — non semplice testo incollato. Il limite per file è intorno ai **30 MB**.

I formati tipici: **.docx** (Word), **.xlsx** (Excel), **.pptx** (PowerPoint), **.pdf**. La regola per scegliere è il destinatario: un report da rileggere e modificare è un .docx; dati e calcoli sono un .xlsx; qualcosa da presentare è un .pptx; un documento finale da distribuire così com’è è un .pdf.

Chiederli bene

Un documento è buono quanto la richiesta. Tre cose alzano la qualità del risultato:

- **Il destinatario e lo scopo:** “una nota per il consiglio”, “un volantino per i clienti”. Cambia tono, lunghezza, struttura.
- **La struttura che vuoi:** le sezioni, l’ordine, cosa mettere in evidenza.
- **Il formato finale:** dì esplicitamente .docx, .xlsx, .pptx o .pdf.

Confronta:

- **Debole:** “Fammi un documento sulle vendite.”
- **Meglio:** “Crea un report .docx di una pagina sulle vendite del Q1 per la direzione: una sintesi iniziale, una tabella per regione, tre punti di commento. Tono sobrio.”

Prima il contenuto, poi il file

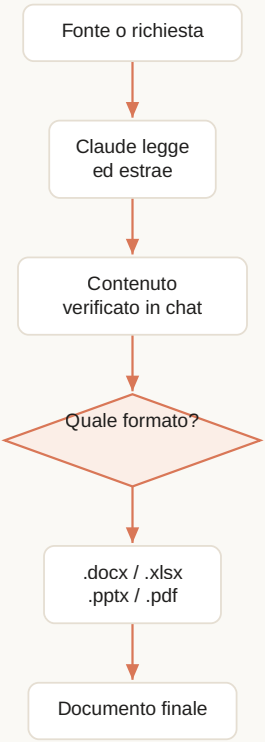
Il principio che fa la differenza è “**read before create**”: prima si fissa il contenuto, poi lo si impagina. Un documento nasce male se chiedi contemporaneamente “trova i dati e fammi il PDF”: il rischio è una bella impaginazione su contenuti sbagliati.

Conviene separare i due momenti:

1. **Contenuto.** Fai produrre o verificare a Claude i dati, i testi, i numeri. Controllali in chat.
2. **Documento.** Solo quando il contenuto è giusto, chiedi di metterlo nel file con la formattazione voluta.

Se parti da un file esistente — un report da riassumere, dati da rielaborare — vale lo stesso: prima Claude **legge** la fonte ed estrae ciò che serve, poi costruisce il nuovo documento. Impaginare per primo significa rifare il lavoro.

Figura L3.4.1 — Il flusso “prima il contenuto, poi il file”. Alt-text: diagramma verticale dal contenuto verificato alla scelta del formato fino al documento finale.



Partire da un modello

Se produci lo stesso tipo di documento spesso — un report mensile, un’offerta — non ripartire da zero ogni volta. Dai a Claude un **modello**: un file di esempio già impostato, o una descrizione precisa della struttura. Lui ci versa dentro il contenuto nuovo mantenendo forma e stile.

È il ponte naturale verso i Project (cap. L3.2): metti il modello nella knowledge base e ogni documento esce coerente con il precedente.

Per la voce e le regole di stile ricorrenti, le Skills (Livello 5) rendono tutto questo ancora più ripetibile.

In pratica: un report in due tempi

1. In chat, fai preparare e **controlla il contenuto**:

   ```
   Dai dati che ti incollo, scrivi la sintesi e
   la tabella per regione. Mostrameli, non fare
   ancora il file.
   ```

2. Correggi quel che serve finché il contenuto è giusto.
3. Chiedi il documento, con formato e struttura:

   ```
   Ora mettilo in un .docx di una pagina: titolo,
   sintesi, tabella, tre punti di commento.
   ```

4. Apri il file e verifica. Per i ritocchi, chiedi modifiche puntuali invece di rigenerare tutto.

Errori comuni

- **Chiedere dati e impaginazione insieme.** Separa: prima il contenuto giusto, poi il file.
- **Non dire il formato.** Senza “.docx” o “.xlsx” rischi un output generico. Sii esplicito.
- **Rigenerare per ogni ritocco.** Chiedi modifiche mirate (“cambia solo la tabella”): più veloce e più sicuro.
- **Dimenticare la sandbox.** Se la creazione file non parte, controlla che la Code execution sia attiva (cap. L1.3).

Riepilogo

1. Claude genera file reali (.docx, .xlsx, .pptx, .pdf) tramite la sandbox di esecuzione codice.
2. Scegli il formato in base al destinatario: report, dati, presentazione, documento finale.
3. Chiedi bene: destinatario, struttura, formato esplicito.
4. “Prima il contenuto, poi il file”: verifica i contenuti in chat, poi impagina.
5. Per documenti ricorrenti parti da un modello; con i Project diventa ripetibile.

Prossimo passo

Nel **cap. L3.5 — Slide ed Excel** entriamo nei due casi più richiesti: costruire una presentazione e un foglio di calcolo con un metodo che evita di rifare il lavoro, e l’uso degli add-in dentro Office.

Dati su Code execution / creazione file (piani, limite ~30 MB) dal ledger, verificati il 22/06/2026 su support.claude.com. Gli esempi non sono stati eseguiti in questa sede.

Capitolo L3.5 — Slide ed Excel

Livello 3 — Lavoro quotidiano. Dati di prodotto verificati il 22/06/2026 su fonti ufficiali.

Obiettivo

Al termine saprai costruire una presentazione con un metodo a tre tempi (struttura, contenuti, stile), impostare un foglio di calcolo passando da dati a formule a grafici, e capire quando conviene l'add-in dentro Office invece della generazione in chat.

Prerequisiti

- Aver letto come si chiedono i documenti (cap. L3.4).
- Code execution / file creation attiva (vedi cap. L1.3).

Slide: struttura, poi contenuti, poi stile

Una presentazione fatta tutta in un colpo viene quasi sempre confusa. Il metodo che funziona è in tre tempi, e segue lo stesso principio "prima il contenuto" del capitolo precedente:

1. **Struttura.** Decidi prima la sequenza delle slide: quante, con quale titolo, in che ordine. È l'ossatura del discorso. Falla rivedere finché il filo regge.
2. **Contenuti.** Riempi ogni slide: pochi punti per slide, un'idea per slide. Qui conta cosa dici, non come appare.
3. **Stile.** Solo alla fine, l'aspetto: tema, colori, coerenza visiva.

Invertire l'ordine costa caro: rifinire lo stile di slide che poi riscrivi è lavoro buttato. Esempio di prima mossa:

```
Prepara la SCALETTA di una presentazione di 8
slide sul progetto X per la direzione: solo
titoli e una riga di contenuto per slide.
```

Quando la scaletta tiene, passi ai contenuti slide per slide, e per ultimo chiedi il `.pptx` con il tema.

Excel: dati, poi formule, poi grafici

Per i fogli di calcolo l'ordine giusto è altrettanto netto:

1. **Dati.** Prima i dati puliti e ben strutturati: colonne con intestazioni chiare, una riga per record. Tutto il resto poggia qui.
2. **Formule.** Poi i calcoli: totali, percentuali, aggregazioni. Su dati ben strutturati le formule vengono semplici e corrette.
3. **Grafici.** Infine la visualizzazione, costruita sui dati e sui calcoli già a posto.

Se i dati sono disordinati, formule e grafici erediteranno il disordine. Vale la pena spendere il primo tempo a sistemare le colonne.

> **Tip:** chiedi a Claude di **spiegare le formule** che inserisce. Un foglio che non capisci è un foglio di cui non puoi fidarti: fartele commentare ti fa da controllo.

Tabella L3.5.1 — L'ordine giusto, a colpo d'occhio.

Output	Prima	Ultimo
Slide	Struttura	Stile
Excel	Dati	Grafici

L’add-in dentro Office

Oltre a generare file in chat, Claude vive **dentro** Office come add-in per **Excel, Word e PowerPoint** (Claude per Microsoft 365). È in beta (research preview) per i piani **Max, Team ed Enterprise**.

La differenza pratica è dove lavori:

- **In chat:** Claude **crea** il file da zero. Comodo quando parti dal nulla o produci molti documenti simili.
- **Nell’add-in:** Claude lavora **sul documento aperto** in Office. Comodo quando il file esiste già, è grande, o vuoi restare nello strumento dove poi lo rifinisci a mano.

Regola spiccia: se il documento nasce ora, la chat va benissimo; se stai modificando qualcosa che è già in Excel o PowerPoint, l’add-in ti evita il viavai tra app.

In pratica: una presentazione in tre tempi

1. Chiedi la **scaletta** (solo titoli + una riga). Aggiustala.
2. Per ogni slide, chiedi i **contenuti**: pochi punti, uno per idea.
3. Chiedi il **.pptx** con un tema coerente:

   ```
   Genera il .pptx dalla scaletta approvata, tema
   sobrio, un colore d'accento, titoli uniformi.
   ```

4. Apri il file. Per i ritocchi, modifiche puntuali (cap. L3.4) o, se ce l’hai, l’add-in di PowerPoint.

Errori comuni

- **Partire dallo stile.** Tema e colori vengono per ultimi. Prima struttura e contenuti.
- **Troppo per slide.** Un'idea per slide. I muri di testo non si presentano.
- **Dati sporchi in Excel.** Formule e grafici ereditano il disordine: sistema prima le colonne.
- **Fidarsi di formule che non capisci.** Fattele spiegare: è il tuo controllo.
- **Cercare l'add-in nel piano sbagliato.** È beta su Max/Team/Enterprise.

Riepilogo

1. Slide in tre tempi: **struttura** → **contenuti** → **stile**. Lo stile per ultimo.
2. Excel in tre tempi: **dati** → **formule** → **grafici**. Dati puliti prima di tutto.
3. Fatti spiegare le formule: un foglio che non capisci non è affidabile.
4. In chat Claude **crea** i file; nell'add-in lavora **sul documento aperto** in Office.
5. L'add-in M365 è beta su Max/Team/Enterprise.

Prossimo passo

Chiuso il lavoro quotidiano, il **Livello 4 — Design** apre il canvas: generare e iterare interfacce, partire dal proprio brand e portare il risultato in Claude Code. Si comincia dal **cap. L4.1 — Design: il canvas**.

Dati sull'add-in Microsoft 365 (prodotti, beta, piani) dal ledger, verificati il 22/06/2026 su claude.com e support.claude.com. Gli esempi non sono stati eseguiti in questa sede.

Capitolo L4.1 — Design: il canvas

Livello 4 — Design. Dati di prodotto verificati il 24/06/2026 su fonti ufficiali.

Obiettivo

Al termine saprai cos'è Claude Design, generare il tuo primo progetto sul canvas conversando, e rifinirlo nei tre modi che il prodotto offre: dalla chat, con i commenti inline e modificando direttamente sulla tela. Imparerai anche a chiedere varianti e a salvare una direzione prima di provarne un'altra.

Prerequisiti

- Saper scrivere richieste efficaci (cap. L1.2): valgono identiche qui.
- Un piano a pagamento: **Pro, Max, Team o Enterprise**. Design è in beta e su Enterprise è **disattivato di default** (lo abilita un admin).

Cos'è Claude Design

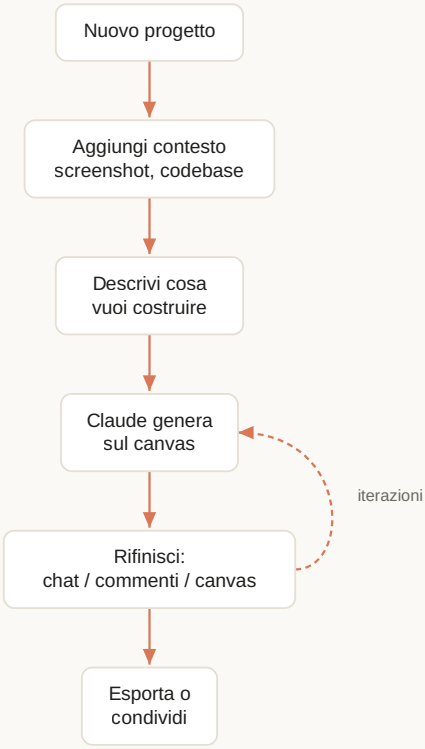
Claude Design è lo strumento per creare interfacce, prototipi interattivi e presentazioni **conversando** con Claude, invece di disegnare a mano. Lo usi sul web (claude.ai/design) o dalla sidebar dell'app desktop; non c'è su mobile.

La schermata ha due aree: la **chat a sinistra**, dove descrivi cosa vuoi, e il **canvas** (la tela) a destra, dove Claude genera il design. Da lì si itera: affini conversando, lasci commenti su punti precisi, o intervieni a mano sulla tela. È lo stesso principio della chat — descrivi il risultato, poi correggi — applicato a un output visuale.

Il flusso di lavoro

Un progetto Design segue quasi sempre lo stesso percorso.

Figura L4.1.1 — Il flusso di un progetto in Claude Design. Alt-text: diagramma verticale dal nuovo progetto alla descrizione, alla generazione sul canvas, alla rifinitura, all'export.



Il progetto eredita in automatico il design system della tua organizzazione, se configurato (lo vediamo nel cap. L4.2): colori, font e componenti sono già al posto giusto, senza caricare nulla.

Scrivere un buon prompt

Non serve essere designer. Serve essere specifici. Un buon prompt dichiara quattro cose:

- **Goal:** cosa stai costruendo.
- **Layout:** come disporre gli elementi.
- **Content:** quali informazioni mostrare.
- **Audience:** chi lo userà.

Per esempio: «Crea una dashboard che mostra il fatturato mensile, con filtri per regione e linea di prodotto». Se manca qualcosa di importante, Claude fa domande prima di generare.

Tre modi per rifinire

La prima generazione è un punto di partenza. Il valore vero è nell'iterazione, e il prodotto offre tre strade — usarle al momento giusto fa risparmiare tempo.

Tabella L4.1.1 — Quale strumento per quale modifica.

Strumento	Per cosa	Esempio
Chat	cambi ampi, strutturali	"tema più scuro"
Commenti inline	mirati, su un componente	"padding più largo"
Modifica diretta	ritocchi visivi rapidi	trascina, ridimensiona

La **chat** è per i cambi che toccano l'insieme o richiedono una spiegazione ("riorganizza la dashboard", "aggiungi un pannello a destra"). I **commenti inline** si applicano cliccando direttamente sull'elemento: più rapidi che descrivere a parole dov'è. La **modifica diretta** sul canvas serve per spostare, ridimensionare e allineare al volo.

> **Attenzione:** a volte un commento inline sparisce prima che Claude lo legga. È un problema noto: se la modifica non viene raccolta, incolla lo stesso feedback nella chat.

Varianti e versioni

Due mosse utili quando non hai ancora deciso la direzione. Per esplorare alternative, chiedi: «Mostrami 2-3 layout diversi per questa pagina»: confrontare è più veloce che indovinare. Per cambiare strada senza perdere il lavoro fatto, di' a Claude: «Salva quello che abbiamo e prova un approccio completamente diverso». Claude conserva la versione attuale e ti conferma dove l'ha salvata, così puoi tornarci.

In pratica: il tuo primo progetto

1. Apri **claude.ai/design** o la sidebar Design nell'app, e crea un progetto.
2. Aggiungi contesto se ne hai: uno screenshot di riferimento, un codebase.
3. Scrivi il prompt con **goal, layout, content, audience**:

```
Landing page per il nostro nuovo prodotto API:
hero, esempi di codice, sezione prezzi.
Pubblico: sviluppatori.
```

4. Guarda cosa genera sul canvas.
5. Rifinisci: chat per i cambi grossi, commenti per i dettagli, mano libera per gli allineamenti.
6. Chiedi 2-3 varianti del layout e scegli da lì.

Errori comuni

- **Prompt vago.** “Fammi un sito” produce risultati generici. Dichiara goal, layout, content, audience.
- **Tutto dalla chat.** Per spostare un elemento o cambiare un padding, commenti inline e modifica diretta sono più rapidi.
- **Feedback non raccolto.** Se un commento inline sparisce, incollalo in chat.
- **Cercare Design su mobile.** È solo web e desktop.

Riepilogo

1. Claude Design crea UI, prototipi e slide **conversando**; chat a sinistra, canvas a destra. Solo web e desktop.
2. Il flusso: progetto → contesto → prompt → generazione → rifinitura → export.
3. Un buon prompt dichiara **goal, layout, content, audience**.
4. Rifinisci con **chat** (cambi ampi), **commenti inline** (mirati), **modifica diretta** (ritocchi visivi).
5. Chiedi **varianti** e **salva una versione** prima di cambiare direzione.

Prossimo passo

Nel **cap. L4.2 — Design system import** vediamo come far partire Design dal tuo brand reale: importare colori, font e componenti da un codebase o da un deck, e ridurre il rischio di un output anonimo.

Dati su Claude Design (aree, flusso, rifinitura, beta/piani) verificati il 24/06/2026 su support.claude.com/en/articles/14604416. Il canvas richiede un account a pagamento, quindi i passaggi non sono stati eseguiti in questa sede.

Capitolo L4.2 — Design system import

Livello 4 — Design. Dati di prodotto verificati il 24/06/2026 su fonti ufficiali.

Obiettivo

Al termine saprai perché far partire Claude Design dal tuo brand invece che da zero, da quali fonti importare un design system, cosa Claude ne estrae, e come evitare l'output anonimo e generico — il cosiddetto "AI slop".

Prerequisiti

- Aver usato il canvas (cap. L4.1).
- Avere almeno una fonte del tuo brand: un codebase, un deck, o asset grafici.

Perché importare il brand

Senza un design system, Claude genera con uno stile predefinito: corretto ma neutro, riconoscibile come "fatto dall'IA". È il problema dell'**AI slop**: risultati plausibili ma senza identità, che sembrano tutti uguali. La cura è dare a Claude il tuo brand come fondamenta, così ogni schermata nasce con i tuoi colori, font e componenti invece che con quelli di default.

Importare un design system è il passo che trasforma Design da generatore di bozze generiche a strumento che produce materiale già "tuo".

Cosa Claude estrae e da dove

Claude analizza le fonti che gli dai ed estrae un **design system riutilizzabile**:

- **Color palette:** colori primari, secondari e d'accento.
- **Tipografia:** famiglie di font, dimensioni, pesi.
- **Componenti:** bottoni, card, elementi di navigazione e altri pattern.
- **Layout:** spaziature, griglie, strutture di pagina.

Le fonti possibili sono più di una, e basta partire da una sola:

Tabella L4.2.1 — Fonti da cui importare e cosa danno.

Fonte	Cosa contiene	Resa
Codebase	componenti reali	la più fedele
Deck / PDF	colori, layout	buona per il look
Asset di brand	logo, palette	base minima

Un **codebase** (per esempio una libreria di componenti React) è la fonte più ricca: Claude legge i componenti veri e li riusa. Un **deck** o un PDF ben fatti bastano a estrarre colori e impaginazione. Anche solo logo e palette danno un punto di partenza. Più fonti dai, più Claude ha da cui lavorare.

Come si imposta

Configurare il design system è un'operazione da **brand owner o designer**, da fare **una volta**: dopo, i progetti del team lo ereditano in automatico (Team/Enterprise). I passaggi, in breve:

1. In Claude Design, seleziona o crea la tua **organizzazione**.
2. Carica gli **asset** del brand (codebase, deck, asset grafici).
3. Claude genera una **UI kit**: la rivedi creando un progetto di prova.

4. Quando ti soddisfa, attiva il **toggle “Published”**: da lì i nuovi progetti dell’org usano il tuo design system.

> **Nota:** in team, il ruolo **Claude Design Admin** permette di approvare un design system standard e bloccare le modifiche, così l’output resta sempre dentro le linee guida aziendali.

Importare da Claude Code

Se il tuo design system vive nel codice, puoi importarlo **dal terminale** con il comando `/design-sync` di Claude Code: porta il design system del codebase locale dentro Design, così ogni schermata parte dai tuoi componenti reali. È il ponte tra codice e canvas, che approfondiamo nel cap. L4.3.

Ridurre l’AI slop

L’import vale quanto la sua fonte: una codebase disordinata o un file incompleto si vedono nell’output. Alcune mosse alzano la qualità:

- **Dai esempi reali, non solo specifiche.** Una landing page finita dice più di una palette di colori isolata: mostra il “sentire” del brand.
- **Itera sull’estrazione.** Se la prima resa non coglie il brand, carica asset diversi o aggiuntivi.
- **Nomina i componenti nel prompt.** «Usa il componente Primary Button» o «Applica il pattern Card» guida Claude verso il tuo sistema invece che verso un default.

In pratica: porta il tuo brand

1. Raccogli la fonte migliore che hai: idealmente il codebase, altrimenti un deck curato.
2. Crea/seleziona l’organizzazione in Claude Design e carica gli asset.

3. Rivedi la UI kit generata con un progetto di test (“Crea una landing page per il nostro prodotto”).
4. Se non coglie il brand, aggiungi asset e itera.
5. Attiva **Published** e verifica che i nuovi progetti usino il tuo stile.

Errori comuni

- **Partire senza design system.** Ottieni l’AI slop: stile anonimo. Importa il brand prima.
- **Fonte disordinata.** Codebase confusa o file incompleto si riflettono nell’output: pulisci la fonte o scegline una migliore.
- **Solo specifiche, niente esempi.** Una palette da sola dice poco: aggiungi una pagina reale.
- **Non nominare i componenti.** Se sai che un componente esiste, chiamalo per nome nel prompt.

Riepilogo

1. Senza design system l’output è anonimo (**AI slop**); importare il brand è la cura.
2. Claude estrae **colori, tipografia, componenti e layout** dalle fonti.
3. La fonte più fedele è un **codebase**; deck e asset danno una base.
4. Il setup è da brand owner, **una volta**: il team eredita il sistema.
5. L’import vale quanto la fonte: dai esempi reali e itera per ridurre l’AI slop.

Prossimo passo

Nel **cap. L4.3 — Da Design a codice** percorriamo il ponte completo: il comando `/design-sync` nei due versi e l’handoff a Claude Code che costruisce software vero a partire dal canvas.

Dati sull'import del design system (fonti, estrazione, setup, ruolo admin) verificati il 24/06/2026 su support.claude.com/en/articles/14604397. Il setup richiede un account a pagamento e non è stato eseguito in questa sede.

Capitolo L4.3 — Da Design a codice (/design-sync)

Livello 4 — Design. Dati di prodotto verificati il 24/06/2026 su fonti ufficiali.

Obiettivo

Al termine capirai il ponte tra Claude Design e Claude Code: come `/design-sync` collega canvas e codice nei due versi, come passare un design a Claude Code perché diventi software vero senza ripartire da uno screenshot, e come avviare un progetto Design direttamente dal terminale. È il capitolo che fa di Design e Code un ciclo unico invece di due strumenti separati.

Prerequisiti

- Claude Code installato e configurato (cap. L2.2, L2.4).
- Aver usato il canvas (cap. L4.1) e, idealmente, importato un design system (cap. L4.2).

Il problema che risolve

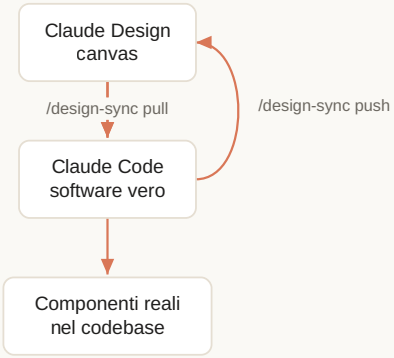
Il punto dolente classico del lavoro design→sviluppo è il salto: il designer consegna uno screenshot o un file, e chi sviluppa **ricostruisce tutto da capo**. Si perde tempo e fedeltà. Il ponte tra Design e Code elimina questo passaggio: Claude Code continua dal lavoro esistente, non da una foto del risultato.

Il ciclo Design ↔ Code

Design e Code non sono una catena a senso unico, ma un anello. Il desi-

gn genera codice; il codice, evolvendo, aggiorna il design. `/design-sync` tiene allineati i due lati.

Figura L4.3.1 — Il ciclo a due vie tra Design e Code. Alt-text: diagramma verticale che mostra il canvas e il codice collegati da pull e push, con l'handoff verso Claude Code.



/design-sync nei due versi

Il comando `/design-sync` si esegue **dentro Claude Code** e funziona in due direzioni.

- **Pull (codice → canvas):** importa il design system del tuo codebase locale in Claude Design, così ogni schermata che generi parte dai tuoi componenti reali.
- **Push (canvas → codice):** dopo che hai implementato un design nel codice, `/design-sync` rimanda nel canvas lo stato attuale, tenendo Design allineato a ciò che hai davvero costruito.

In pratica: parti dal codice per dare a Design le fondamenta giuste (pull), costruisci e iteri, poi rimandi indietro il risultato per non far divergere canvas e prodotto (push).

Tabella L4.3.1 — Le due direzioni di `/design-sync`.

Verso	Da → A	A cosa serve
Pull	codice → canvas	design su componenti reali
Push	canvas → codice	canvas allineato al build

L'handoff a Claude Code

Quando un design è pronto a diventare software, lo **passi** a Claude Code (handoff). Code continua dal lavoro esistente invece di ricostruirlo da uno screenshot. L'handoff si avvia anche dal pulsante **Export** di Design (vedi cap. L4.5) e può puntare al coding agent locale o a Claude Code Web.

Partire da Claude Code: /design

Se preferisci restare nel terminale, non sei obbligato ad aprire il canvas. Con il comando `/design` avvii, modifichi e sincronizzi un progetto Design **senza lasciare Claude Code**: importi un design nel codebase, esporti il codice come prototipo live, o lasci che Claude costruisca tutto da capo.

> **Nota:** se in Claude Code non vedi `/design` o `/design-sync`, esegui `/update` per aggiornare le skill. I comandi compaiono solo nelle **sessioni nuove**, non in quella già aperta.

In pratica: dal codebase al canvas e ritorno

1. In Claude Code, dentro il tuo progetto, esegui:

```
/design-sync
```

2. Scegli **pull**: importi il design system del codebase in Design.
3. In Design, genera e itera le schermate: useranno i tuoi componenti reali.
4. Quando una schermata è pronta, fai **handoff** a Claude Code e costruisci.
5. Dopo l'implementazione, di nuovo `/design-sync` in **push** per riallineare il canvas al codice.

> **Tip:** se i comandi non compaiono, `/update` e poi apri una sessione nuova.

Errori comuni

- **Aspettarsi i comandi nella sessione corrente.** Dopo `/update`, `/design` e `/design-sync` ci sono solo nelle sessioni nuove.
- **Handoff da screenshot.** Non serve: l'handoff porta il lavoro reale, non una foto. Passa il progetto, non un'immagine.
- **Canvas e codice che divergono.** Dopo aver costruito, ricordati il **push**: senza, Design resta indietro rispetto al prodotto.
- **Saltare il pull iniziale.** Senza, Design genera con lo stile di default e non con i tuoi componenti.

Riepilogo

1. Il ponte Design ↔ Code elimina il "ricostruisci da screenshot".
2. `/design-sync` gira in Claude Code in due versi: **pull** (codice → canvas) e **push** (canvas → codice).
3. L'**handoff** passa il design a Code, che continua dal lavoro reale.
4. Con `/design` avvii e sincronizzi un progetto Design **dal terminale**.
5. Se i comandi mancano, `/update` e apri una **sessione nuova**.

Prossimo passo

Nel **cap. L4.4 — Design dentro Cowork** vediamo come usare gli output di Design come materiale per i task agentici, e come le Skills rendono ripetibile il lavoro visuale.

Dati su `/design-sync`, handoff e `/design` verificati il 24/06/2026 su support.claude.com/en/articles/14604416. I comandi richiedono Claude Code con un account a pagamento, quindi non sono stati eseguiti in questa sede.

Capitolo L4.4 — Design dentro Cowork

Livello 4 — Design. Concetti di composizione tra prodotti; dettagli verificati il 24/06/2026.

Obiettivo

Al termine saprai usare ciò che produci in Claude Design come **materiale** per i task agentici di Cowork, e capirai come le Skills rendono il lavoro visuale ripetibile invece di doverlo rispiegare ogni volta. È un capitolo di composizione: non un nuovo strumento, ma due che già conosci messi a lavorare insieme.

Prerequisiti

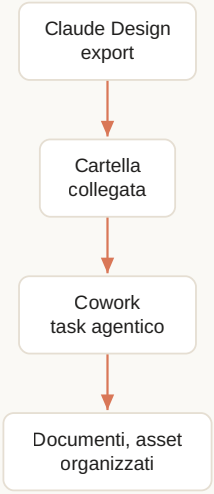
- Saper avviare un task in Cowork (cap. L3.1).
- Aver prodotto qualcosa in Design (cap. L4.1) ed esportato un output (cap. L4.5).

Design produce, Cowork lavora

Design e Cowork rispondono a bisogni diversi. Design **crea** materiale visuale: schermate, prototipi, slide, codice di un prototipo. Cowork **agisce** su una cartella di file in più passi. Il punto di contatto è naturale: gli output di Design diventano gli **input** di Cowork.

Il collegamento passa dalla cartella. Esporti da Design (per esempio uno ZIP, un HTML standalone o il codice di un prototipo, vedi cap. L4.5), lo metti in una cartella collegata a Cowork, e da lì Cowork ci lavora: riorganizza gli asset, genera la documentazione, prepara i file per la consegna. Ricorda che Cowork vede **solo le cartelle che colleghi** (cap. L3.1): l'asset esiste per lui solo dopo che l'hai messo lì.

Figura L4.4.1 — Dagli output di Design ai task di Cowork. Alt-text: diagramma verticale dall'export di Design alla cartella collegata fino al task agentico di Cowork.



Le Skills rendono il design ripetibile

Il vero salto di qualità non è spostare un file: è rendere **ripetibile** il modo in cui il lavoro visuale viene trattato. Qui entrano le Skills (Livello 5): una skill scrive una volta le regole — come nominare gli asset, quale struttura di cartelle usare, quali controlli fare su un export — e Claude le applica allo stesso modo in chat, in Cowork e in Code.

Senza skill, ripeti le istruzioni a ogni task ("metti gli asset in `/img`, rinomina così, genera l'indice"). Con una skill, le dici una volta e valgono sempre. È la differenza tra un risultato che dipende da quanto sei stato preciso oggi e un risultato costante.

Un esempio concreto

Immagina di produrre in Design i mockup di una nuova sezione del sito e di esportarli come HTML standalone. Il flusso composto:

1. Esporti i mockup da Design in una cartella di progetto.
2. Colleghi la cartella a Cowork.
3. Dai a Cowork un task come **end-state** (cap. L3.1): «Organizza questi mockup per pagina, genera un `INDICE.md` con anteprime e note, e prepara una cartella `consegna/` pronta da inviare».
4. Una skill di progetto garantisce che nomi, struttura e controlli siano sempre gli stessi, mockup dopo mockup.

Il risultato: Design fa la parte creativa, Cowork la parte ripetitiva e organizzativa, e la skill tiene insieme le regole.

In pratica: comporre Design e Cowork

1. In Design, esporta l'output che ti serve (cap. L4.5) in una cartella.
2. In Cowork, **collega** quella cartella.
3. Scrivi il task come risultato voluto, non come sequenza di passi.
4. Se ripeti lo stesso tipo di lavoro, racchiudi le regole in una **skill** di progetto (Livello 5).
5. Rilancia su nuovi asset: la skill mantiene la coerenza.

Errori comuni

- **Aspettarsi un collegamento automatico.** Design non "entra" in Cowork da solo: passi per un export e una cartella collegata.
- **Dimenticare di collegare la cartella.** Cowork vede solo ciò che colleghi: se l'asset non è lì, per lui non esiste (cap. L3.1).
- **Ripetere le regole a ogni task.** Se il lavoro ricorre, mettilo in una skill invece di rispiegarlo.

- **Guidare Cowork passo per passo.** Vale anche qui: descrivi l'end-state.

Riepilogo

1. Design **crea** materiale visuale; Cowork **agisce** sui file: gli output del primo sono input del secondo.
2. Il collegamento passa da un **export** in una **cartella collegata** a Cowork.
3. Le **Skills** rendono ripetibile il trattamento del lavoro visuale, uguale in chat, Cowork e Code.
4. Schema tipico: Design crea, Cowork organizza, la skill tiene le regole.
5. In Cowork descrivi sempre il **risultato**, non i passi.

Prossimo passo

Nel **cap. L4.5 — Export e Canva** chiudiamo il Livello 4 con i formati di uscita di Design — PDF, PPTX, HTML, Canva e gli altri — e con il criterio per scegliere tra esportare e fare handoff.

Capitolo di composizione tra Design (cap. L4.1, L4.5) e Cowork (cap. L3.1). Le capacità delle Skills sono introdotte qui e approfondite al Livello 5. Nessun comando eseguito in questa sede.

Capitolo L4.5 — Export e Canva

Livello 4 — Design. Dati di prodotto verificati il 24/06/2026 su fonti ufficiali.

Obiettivo

Al termine saprai esportare un progetto di Claude Design nel formato giusto, mandarlo agli strumenti che già usi (Canva e altri), e scegliere con criterio tra **esportare** un file e fare **handoff** verso il codice. È il capitolo che porta il lavoro fuori dal canvas.

Prerequisiti

- Un progetto pronto sul canvas (cap. L4.1).
- Per l'handoff, Claude Code installato (cap. L2.2) e il concetto di ponte Design ↔ Code (cap. L4.3).

Dove sta l'export

Quando il design è pronto, il pulsante **Export** in alto a destra apre le vie di uscita. La scelta del formato dipende da cosa devi farne: raccogliere un parere, consegnare allo sviluppo, o presentare a un gruppo.

I formati di uscita

Le destinazioni si raggruppano per scopo. La tabella riassume le principali.

Tabella L4.5.1 — Formati di export e quando usarli.

Scopo	Formato	Quando
Rivedere	PDF	feedback, stampa
Presentare	PPTX / Canva	slide da rifinire
Pubblicare	HTML / ZIP	prototipo, web

Oltre a questi, Design esporta come **.zip**, manda **a Canva** e produce **HTML standalone**. Può inoltre inviare il progetto a strumenti esterni — Adobe, Base44, Canva, Gamma, Lovable, Miro, Replit, Vercel, Wix — con altre destinazioni in arrivo. Per la condivisione interna all'organizzazione c'è un **link** con permessi a tre livelli: sola visione, commento, modifica.

Canva e gli strumenti esterni

"Send to Canva" porta il design dentro Canva, dove lo rifinisci con strumenti di impaginazione che già conosci. La stessa logica vale per le altre destinazioni: Design non vuole sostituire i tuoi strumenti, ma **consegnare** loro un punto di partenza già impostato sul tuo brand, invece di farti ricominciare da una pagina bianca.

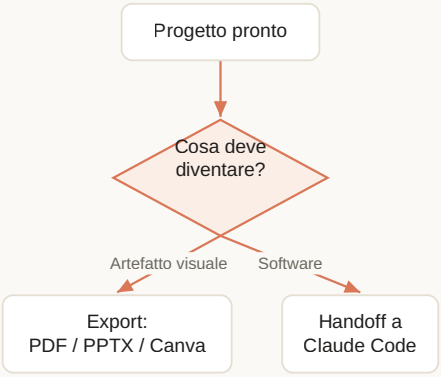
Esportare o fare handoff?

È la scelta che conta di più, e dipende dalla destinazione del lavoro.

- **Esporta** quando l'output è il fine: un PDF per un parere, un PPTX da presentare, un file Canva da rifinire. Il design resta un artefatto visuale.
- **Fai handoff** quando il design deve diventare **software**. L'handoff a Claude Code (cap. L4.3) continua dal lavoro reale, senza ricostruire da uno screenshot, e punta al coding agent locale o a Claude Code Web.

Regola pratica: se la prossima persona a toccarlo apre un visualizzatore, esporta; se apre un editor di codice, fai handoff.

Figura L4.5.1 — Esportare o fare handoff: il bivio. Alt-text: diagramma verticale che dal progetto pronto separa il ramo export dal ramo handoff verso il codice.



In pratica: porta fuori il tuo design

1. Apri il progetto e premi **Export** in alto a destra.
2. Per un parere: **PDF**. Per presentare: **PPTX** o **Send to Canva**.
3. Per un prototipo web: **HTML standalone** o **.zip**.
4. Se deve diventare software, scegli **Handoff a Claude Code** invece di un file.
5. Per condividere nel team, genera un **link** con il permesso giusto (visione, commento o modifica).

Errori comuni

- **Esportare uno screenshot per lo sviluppo.** Se diventa codice, fai handoff: porti il lavoro reale, non una foto.

- **Formato sbagliato per lo scopo.** PDF per un parere, PPTX/Canva per presentare, HTML/ZIP per il web.
- **Link con permesso troppo ampio.** Per un semplice parere basta “sola visione” o “commento”, non “modifica”.
- **Cercare destinazioni che non ci sono ancora.** L’elenco cresce nel tempo: verifica quali sono disponibili al momento.

Riepilogo

1. Il pulsante **Export** (in alto a destra) apre le vie di uscita di Design.
2. Formati: **PDF, PPTX, HTML, .zip**, invio a **Canva** e ad altri strumenti.
3. **Send to Canva** e simili consegnano un punto di partenza già sul brand.
4. **Esporta** per un artefatto visuale; **handoff** quando deve diventare software.
5. Condividi nell’org con un **link** a permessi: visione, commento, modifica.

Prossimo passo

Hai completato il Livello 4. Nel **Livello 5 — Skills e identità** vediamo cosa sono davvero le Skills — già incontrate come collante di Cowork e Design — a partire dal **cap. L5.1 — Anatomia di una skill**.

Dati su export, destinazioni e condivisione verificati il 24/06/2026 su support.claude.com/en/articles/14604416. Le funzioni richiedono un account a pagamento, quindi non sono state eseguite in questa sede.

Capitolo L5.1 — Anatomia di una skill

Livello 5 — Skills e identità. Dati di prodotto verificati il 24/06/2026 su fonti ufficiali.

Obiettivo

Al termine saprai cos'è una skill, in cosa differisce da un prompt, come è fatto un file `SKILL.md` e perché la sua `description` è la parte più importante. È la base concettuale per costruirne una tua nel prossimo capitolo.

Prerequisiti

- Saper scrivere richieste efficaci (cap. L1.2).
- Avere **Code execution** attiva (cap. L1.3): è richiesta per le Skills.

Skill o prompt?

Un prompt vale per una conversazione: lo scrivi, ottieni la risposta, finisce lì. Una **skill** è conoscenza riutilizzabile che Claude carica **da solo** quando il contesto lo richiede, senza che tu la incolli ogni volta. Scrivi una regola una volta, e Claude la applica ogni volta che serve.

La differenza pratica: il prompt è un'istruzione usa-e-getta; la skill è una competenza permanente. Se ti accorgi di reincollare le stesse istruzioni in chat dopo chat, quel testo è un candidato a diventare una skill.

Una cartella e un file

Una skill, nella sua forma minima, è una **cartella** che contiene un file `SKILL.md`. Solo quel file è obbligatorio: basta lui a fare una skill funzionante. Il file ha due parti:

- un **frontmatter** in cima (metadati tra due righe `---`);
- un **corpo** in Markdown con le istruzioni vere e proprie.

> **Nota:** il file si chiama per convenzione `SKILL.md`. La documentazione lo cita anche come `skill.md`: è lo stesso file. In Claude Code le skill di progetto stanno sotto `.claude/skills/` (cap. L2.4).

Il frontmatter: due campi obbligatori

Il frontmatter dichiara chi è la skill. Due campi sono obbligatori:

- **name** — il nome leggibile, massimo **64 caratteri**.
- **description** — cosa fa la skill e **quando usarla**, massimo **200 caratteri**.

Esistono campi opzionali (per esempio `dependencies` per i pacchetti software), ma name e description bastano per partire.

Ecco una skill minima completa:

```
---
name: verbale-riunione
description: Da note grezze crea un verbale
  con decisioni e action item.
---

# Verbale riunione

Trasforma note disordinate in un verbale:
1. Elenca le decisioni prese.
2. Estrai gli action item con responsabile.
3. Chiudi con i punti rimasti aperti.
```

La description è tutto

Tra i due campi, la `description` è quello che conta davvero. Claude la legge per decidere **se** attivare la skill: con decine o centinaia di skill disponibili, è la description a far scattare quella giusta al momento giusto. Una description vaga (“aiuta con i documenti”) non scatta mai con precisione; una specifica (“da note grezze crea un verbale con decisioni e action item”) sì.

Qui entra il meccanismo della **progressive disclosure** (rivelazione progressiva): Claude legge prima solo i metadati — name e description — per capire se la skill serve, e carica il corpo completo **solo se** decide di usarla. È efficiente: poche righe gli bastano per scegliere, senza leggere tutto. Per questo la description va scritta pensando a quando vuoi che la skill si attivi, non solo a cosa fa.

Il corpo: istruzioni ed esempi

Sotto il frontmatter, il corpo Markdown contiene le istruzioni che Claude segue quando la skill è attiva: cosa fare, in che ordine, con quali vincoli. Gli esempi aiutano molto — mostrare un input e l’output atteso vale più di una spiegazione astratta. Se le istruzioni crescono troppo, puoi

spostarne una parte in file di risorse separati (per esempio un `REFERENCE.md`) e richiamarli dal `SKILL.md`.

Errori comuni

- **Confondere skill e prompt.** Il prompt è per una chat; la skill è riutilizzabile e si attiva da sola.
- **Description vaga.** È il campo che fa scattare la skill: sii specifico sul "quando".
- **Sforare i limiti.** name oltre 64 o description oltre 200 caratteri non sono validi.
- **Mettere tutto nel corpo.** Se è troppo, usa file di risorse separati.

Riepilogo

1. Una skill è conoscenza **riutilizzabile** che Claude carica da solo; un prompt vale per una sola chat.
2. Nella forma minima è una **cartella** con un file `SKILL.md`.
3. Il frontmatter ha due campi obbligatori: **name** (≤64) e **description** (≤200).
4. La **description** decide quando la skill scatta: scrivila pensando al "quando".
5. Il corpo Markdown contiene istruzioni ed esempi; le risorse extra vanno in file separati.

Prossimo passo

Nel **cap. L5.2 — La tua prima skill** costruiamo una skill da zero, a mano e con l'aiuto di skill-creator, e impariamo a testare che si attivi quando deve.

Dati su SKILL.md, frontmatter e limiti verificati il 24/06/2026 su support.claude.com/en/articles/12512198 e code.claude.com/docs/en/skills. L'esempio di SKILL.md non è stato eseguito in questa sede.

Capitolo L5.2 — La tua prima skill

Livello 5 — Skills e identità. Dati di prodotto verificati il 24/06/2026 su fonti ufficiali.

Obiettivo

Al termine avrai creato una skill funzionante: la struttura della cartella, una `description` che “scatta” quando deve, e il modo di testarla. Vedremo sia la via manuale sia l’aiuto di **skill-creator**, la skill che assiste a costruirne altre.

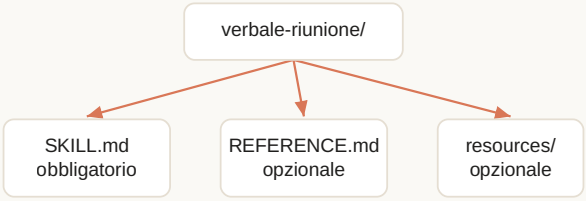
Prerequisiti

- Capire l’anatomia di una skill (cap. L5.1).
- **Code execution** attiva (cap. L1.3).

La struttura della cartella

Una skill è una cartella il cui nome coincide con il nome della skill, con dentro almeno il file `SKILL.md`. Se aggiungi risorse o script, stanno nella stessa cartella.

Figura L5.2.1 — Struttura di una cartella skill. Alt-text: diagramma verticale della cartella skill con SKILL.md e una sottocartella di risorse.



Per usarla in chat o in Cowork, la cartella va compressa in uno **ZIP** con la cartella stessa come radice (non i file sciolti nello ZIP), e poi caricata. In Claude Code, invece, la metti sotto `.claude/skills/` del progetto (cap. L2.4).

Scrivere una description che scatta

È il passo che decide tutto. La `description` deve dire **cosa** fa la skill e **quando** Claude deve usarla, in massimo 200 caratteri. Due esempi a confronto:

- **Debole:** “Aiuta con le email.” — troppo generica, scatta a caso o mai.
- **Forte:** “Scrive risposte email cortesi e brevi a clienti, a partire dal messaggio ricevuto.” — dice il compito e il contesto d’uso.

La regola: includi i segnali che vuoi facciano scattare la skill. Se deve attivarsi sui verbali, nomina i verbali; se sui clienti, nomina i clienti.

Costruirla con skill-creator

Non sei costretto a partire dal foglio bianco. **skill-creator** è una skill ufficiale che ti guida a crearne una: imposta la struttura, ti aiuta a scrivere la description e può generare prompt di prova (eval) per controllare che la skill scatti quando deve — e, importante, che **non** scatti quando non deve.

È il modo più rapido per una prima skill solida: descrivi il workflow che vuoi automatizzare, e skill-creator produce lo scheletro che poi rifinisci.

Testare prima e dopo l’upload

Una skill si valuta sul campo. Prima di caricarla, rileggi il `SKILL.md`, controlla che la description rifletta davvero quando serve, e verifica che

i file richiamati esistano. Dopo averla caricata e abilitata in **Customize > Skills**, provala con più prompt che dovrebbero attivarla, e controlla nel "thinking" di Claude che la stia caricando. Se non scatta quando te lo aspetti, il punto da correggere è quasi sempre la **description**.

In pratica: crea e prova una skill

1. Crea una cartella con il nome della skill (es. `verbale-riunione`).
2. Dentro, scrivi `SKILL.md` con frontmatter (name, description) e corpo.
3. Cura la **description**: cosa fa e quando usarla, sotto i 200 caratteri.
4. Per la chat/Cowork, comprimi la cartella in **ZIP** (cartella come radice) e caricala; abilitala in **Customize > Skills**.
5. Prova con prompt che devono attivarla; controlla che si carichi.
6. Se non scatta, **itera sulla description** e riprova.

Errori comuni

- **ZIP con i file sciolti.** Lo ZIP deve contenere la **cartella** come radice, non i file direttamente.
- **Nome cartella diverso dal nome skill.** Falli coincidere.
- **Description che non scatta.** Aggiungi i segnali del "quando"; non descrivere solo il "cosa".
- **Provarla una volta sola.** Testa con più prompt, inclusi alcuni che **non** devono attivarla.
- **Hardcodare segreti.** Mai chiavi o password dentro una skill.

Riepilogo

1. Una skill è una **cartella** (nome = nome skill) con dentro `SKILL.md`.
2. Per chat/Cowork si carica come **ZIP** con la cartella come radice; in Code va in `.claude/skills/`.

3. La **description** che scatta nomina cosa fa **e** quando usarla.
4. **skill-creator** scaffolda la skill e genera prove di attivazione.
5. Testa prima e dopo l’upload; se non scatta, itera sulla description.

Prossimo passo

Nel **cap. L5.3 — Skills operative in Cowork** vediamo le Skills al lavoro nei task autonomi: differenze tra chat e Cowork, come spezzarle, e la condivisione in team. Useremo come esempio le tre skill con cui è scritto questo libro.

Dati su struttura, packaging ZIP, test e skill-creator verificati il 24/06/2026 su support.claude.com/en/articles/12512198. Nessuna skill è stata creata o eseguita in questa sede.

Capitolo L5.3 — Skills operative in Cowork

Livello 5 — Skills e identità. Dati di prodotto verificati il 24/06/2026 su fonti ufficiali.

Obiettivo

Al termine saprai usare le Skills nel lavoro autonomo: come si comportano in chat rispetto a Cowork, perché conviene spezzarle in più skill focalizzate invece di una sola grande, e come condividerle in team. L'esempio guida sono le tre skill con cui è scritto questo libro.

Prerequisiti

- Saper creare una skill (cap. L5.2).
- Conoscere Cowork (cap. L3.1).

In chat e in Cowork

Una stessa skill vale in chat, in Cowork e in Claude Code: la scrivi una volta e guida Claude allo stesso modo ovunque. Cambia il contesto in cui lavora.

In **chat** la skill rifinisce una singola risposta: tono, formato, struttura. In **Cowork**, dove Claude esegue un task lungo su una cartella in più passi, la skill conta di più: governa un comportamento ripetuto su molti file, senza che tu debba ripetere le regole a ogni passaggio. Più il lavoro è autonomo e lungo, più una skill ben scritta fa la differenza tra un risultato costante e uno che dipende da quanto sei stato preciso oggi.

Spezzare invece di accumulare

La tentazione è scrivere una skill che fa tutto. È quasi sempre un errore. Più skill **focalizzate** compongono meglio di una monolitica, per due ragioni: la `description` di una skill mirata scatta con precisione (cap. L5.1), mentre quella di una skill tuttofare è per forza vaga; e Claude può **combinare** più skill in automatico quando servono insieme.

Regola pratica: una skill, un workflow. Se ti accorgi che una skill ha due scopi distinti, dividila. Le skill non possono richiamarsi a vicenda, ma Claude ne usa diverse nello stesso task: la composizione la fa lui.

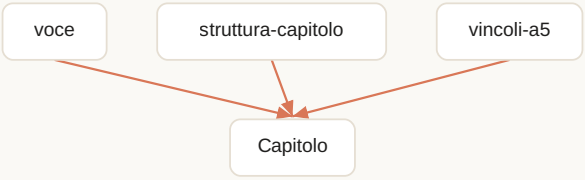
Esempio: le tre skill di questo libro

Questo manuale è scritto con tre skill separate invece di un'unica grande:

- **voce** — tono e terminologia: italiano pratico, termini di prodotto in inglese, niente entusiasmo da brochure.
- **struttura-capitolo** — struttura standard, target di lunghezza, uso del ledger dei fatti, definition of done.
- **vincoli-a5** — vincoli di impaginazione: codice stretto, tabelle a tre colonne, diagrammi verticali.

Ognuna ha uno scopo netto e una description che scatta sul suo ambito. Quando scrivo un capitolo, Claude le usa **tutte e tre insieme**: la voce regola il testo, la struttura l'ossatura, i vincoli la forma. Una sola skill che facesse il lavoro di tutte e tre sarebbe più difficile da scrivere, da far scattare e da aggiornare.

Figura L5.3.1 — Tre skill focalizzate che Claude compone su un capitolo. Alt-text: diagramma verticale con tre skill che convergono sulla scrittura di un capitolo.



Condividere in team

Una skill utile non deve restare sul tuo account. Sui piani **Team ed Enterprise** un amministratore può fornire e gestire le skill per tutta l'organizzazione, così tutti lavorano con le stesse regole. È il modo per rendere uno standard — di voce, di formato, di processo — qualcosa che vale per il gruppo e non solo per chi l'ha scritto.

> **Tip:** prima di condividere una skill in team, falla provare a qualcun altro su prompt reali. Una description che scatta bene per te può scattare male per chi usa parole diverse.

In pratica: porta le tue skill in Cowork

1. Parti da un workflow che ripeti spesso in Cowork (es. organizzare asset, produrre un report).
2. Scrivine le regole in una skill **focalizzata** (cap. L5.2).
3. Se gli scopi sono due, fai **due** skill, non una.
4. Attivala e lancia un task in Cowork: verifica che le regole siano applicate.
5. Se sei in un team, chiedi all'admin di renderla disponibile all'organizzazione.

Errori comuni

- **La skill tuttofare.** Description vaga e difficile da mantenere. Spezza per workflow.
- **Aspettarsi che una skill ne richiami un'altra.** Non succede: è Claude a comporle. Scrivile autonome.
- **Condividere senza far provare.** In team, una description va testata su più modi di scrivere lo stesso intento.
- **Ripetere le regole nel prompt.** Se sono in una skill, non serve ridirle a ogni task.

Riepilogo

1. La stessa skill vale in chat, Cowork e Code; in Cowork pesa di più perché governa task lunghi e autonomi.
2. Più skill **focalizzate** compongono meglio di una monolitica.
3. Le skill non si richiamano tra loro: la **composizione** la fa Claude.
4. Le tre skill di questo libro (voce, struttura, vincoli) sono l'esempio del pattern.
5. In **Team/Enterprise** un admin condivide le skill con tutta l'organizzazione.

Prossimo passo

Nel **cap. L5.4 — Far suonare Claude come te** mettiamo insieme tutti gli strumenti dell'identità — istruzioni, Projects, file di contesto e Skills — per dare a Claude la tua voce in modo stabile.

Dati su uso, composizione e condivisione delle skill verificati il 24/06/2026 su support.claude.com/en/articles/12512198. L'esempio delle tre skill è reale (sono quelle del progetto di questo libro). Nessuna skill eseguita in questa sede.

Capitolo L5.4 — Far suonare Claude come te

Livello 5 — Skills e identità. Concetti stabili; dettagli di prodotto dal ledger (verificati 24/06/2026).

Obiettivo

Al termine saprai comporre gli strumenti dell'identità — istruzioni d'account, Styles, Projects, file di contesto e Skills — per dare a Claude una voce stabile e riconoscibile, e farai un esercizio pratico per estrarre la tua. È il capitolo che chiude il Livello 5 mettendo in fila ciò che hai imparato.

Prerequisiti

- Impostazioni e Styles (cap. L1.3), Projects (cap. L3.2), Skills (cap. L5.1-5.3).

Perché la voce conta

Claude, senza indicazioni, scrive in un registro corretto ma neutro. Se lo usi per produrre testi che portano il tuo nome — email, articoli, documenti — quel neutro si sente: è il segno che il testo non è "tuo". Dare a Claude la tua voce non è vezzo estetico, è ciò che rende il risultato pubblicabile senza riscriverlo.

La voce non sta in un solo posto. Si compone di più strumenti, ognuno con un raggio d'azione diverso. Sceglierli bene evita di mettere tutto ovunque.

La scala degli strumenti

Dal più leggero al più strutturato, ecco dove vive l'identità e cosa governa.

Tabella L5.4.1 — Gli strumenti dell'identità, dal leggero allo strutturato.

Strumento	Raggio	Per cosa
Instructions	ogni chat	preferenze stabili
Styles	all'occorrenza	come comunicare
Projects	un tema/cliente	contesto isolato
Skills	ovunque serve	regole ripetibili

- **Instructions for Claude** (cap. L1.3): preferenze personali valide in ogni chat — nome, lingua, tono di base. Il livello più comodo per ciò che vale sempre.
- **Styles** (cap. L1.3): regolano *come* Claude comunica; li attivi quando servono, e uno stile custom può imparare dal tuo testo.
- **Projects** (cap. L3.2): isolano contesto e istruzioni per un tema o un cliente, senza inquinare le altre chat.
- **Skills** (cap. L5.1-5.3): impacchettano regole ripetibili che scattano da sole, in chat, Cowork e Code.

I file di contesto

C'è un livello che precede tutti: i **file di contesto**. Sono documenti dove raccogli il materiale grezzo della tua voce — esempi dei tuoi testi, lista di parole da evitare, coppie "prima/dopo". Non sono un prodotto a sé: sono la materia prima da cui poi nascono uno Style custom, le istruzioni di un Project o una skill di voce.

Questo libro funziona così: un file di contesto raccoglie esempi e regole di stile, e da lì si alimentano la skill `voce` e le istruzioni del Project. Scrivere prima il contesto evita di disperdere la voce in tante piccole istruzioni scollegate.

Comporli, non accumularli

L'errore è mettere la stessa regola in tutti gli strumenti. Meglio assegnare a ciascuno il suo raggio: le preferenze sempre valide nelle Instructions; il "come scrivere" negli Styles o in una skill `voce`; il materiale di un cliente in un Project. Una regola sta in un posto solo — quello giusto — così, quando cambia, la aggiorni una volta.

Esercizio: estrai la tua voce

Un percorso concreto per dare a Claude la tua voce:

1. **Raccogli** 3-5 tuoi testi reali (email, articoli, note). Sono il materiale più prezioso.
2. **Fai estrarre i tratti:** chiedi a Claude «Da questi miei testi, descrivi il mio registro, le parole ricorrenti e quelle che evito».
3. **Scrivi il file di contesto:** salva quei tratti, con esempi e una lista di parole da evitare.
4. **Scegli il contenitore:** per un uso generale, uno **Style custom**; per un lavoro ricorrente, una **skill** `voce`; per un cliente, un **Project**.
5. **Prova e correggi:** fai produrre un testo, confrontalo con i tuoi, e affina il contesto finché "suona" come te.

> **Tip:** il test della voce è semplice. Fai scrivere a Claude un breve testo e chiediti: lo firmeresti? Se no, manca un tratto: aggiungilo al contesto.

Errori comuni

- **Lasciare a Claude il registro di default.** Su testi che portano il tuo nome si sente. Dagli la tua voce.
- **Stessa regola ovunque.** Assegna a ogni strumento il suo raggio; una regola, un posto.
- **Descrivere la voce a parole.** Gli esempi reali valgono più di mille aggettivi: parti dai tuoi testi.
- **Fermarsi alla prima resa.** La voce si affina iterando sul file di contesto.

Riepilogo

1. Dare a Claude la tua voce rende i testi pubblicabili senza riscriverli.
2. Gli strumenti hanno raggi diversi: **Instructions** (sempre), **Styles** (come scrivere), **Projects** (un tema), **Skills** (regole ripetibili).
3. I **file di contesto** sono la materia prima da cui nascono Style, Project e skill di voce.
4. **Componi** gli strumenti: una regola in un posto solo.
5. Estrai la voce dai tuoi **testi reali**, poi scegli il contenitore e itera.

Prossimo passo

Hai completato il Livello 5. Nel **Livello 6 — Avanzato** si passa al controllo fine degli strumenti, a partire dal **cap. L6.1 — Claude Code avanzato**: hooks, sub-agent e permessi granulari.

Capitolo di sintesi tra Instructions/Styles (cap. L1.3), Projects (cap. L3.2) e Skills (cap. L5.1-5.3). Dettagli di prodotto dal ledger, verificati il 24/06/2026. Nessun comando eseguito in questa sede.

Capitolo L6.1 — Claude Code avanzato

Livello 6 — Avanzato. Dati di prodotto verificati il 24/06/2026 su fonti ufficiali.

Obiettivo

Al termine conoscerai tre leve avanzate di Claude Code: gli **hooks** per agganciare i tuoi comandi agli eventi, i **sub-agent** per delegare compiti a contesti separati, e i **permessi** granulari. Sono gli strumenti che trasformano Claude Code da assistente a parte integrata del tuo flusso.

Prerequisiti

- Claude Code installato e un progetto configurato (cap. L2.2, L2.4).
- A tuo agio con `settings.json` e i permessi base (cap. L2.4).

Hooks: agganciare comandi agli eventi

Un **hook** è un comando shell che Claude Code esegue automaticamente in corrispondenza di un **evento** del suo ciclo di vita: prima di usare un tool, dopo, all'invio di un prompt, e così via. Servono a imporre regole che non vuoi affidare alla buona volontà: lanciare il linter dopo ogni modifica, bloccare la scrittura su certi file, registrare cosa è successo.

Si configurano in `settings.json`, organizzati per **matcher** (il nome del tool, una regex, o vuoto per tutti). Gli eventi principali sono `PreToolUse` (prima della chiamata a un tool) e `PostToolUse` (dopo).

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit",
        "hooks": [
          { "type": "command",
            "command": "npm run lint" }
        ]
      }
    ]
  }
}
```

La trappola dell'exit code

Il punto che fa inciampare tutti è il **codice di uscita** dell'hook. Non è un dettaglio: cambia il comportamento di Claude.

- **exit 0:** tutto ok, si prosegue.
- **exit 2:** errore **bloccante**. Lo stdout viene ignorato; lo **stderr** torna a Claude come messaggio d'errore. Su `PreToolUse`, la chiamata al tool è bloccata.
- **altri codici != 0:** errore **non** bloccante, mostrato a te.

Quindi, se vuoi che un hook **fermi** un'azione e spieghi a Claude perché, devi uscire con **exit 2** e scrivere il motivo su **stderr**. Sbagliare codice è la causa più comune di hook che "non bloccano".

Sub-agent: delegare a un contesto separato

Un **sub-agent** è un assistente specializzato che Claude Code interpella per un compito, con un **proprio context window**, un proprio system prompt e propri permessi sui tool. Serve a isolare un lavoro (per esempio una revisione del codice) senza ingombrare la conversazione principale.

Si definisce come file Markdown con frontmatter in `.claude/agents/` (di progetto) o `~/.claude/agents/` (personale). Claude decide a quale sub-agent delegare leggendo la `description`, come per le Skills.

```
---
name: code-reviewer
description: Rivede le modifiche per qualità
  e correttezza. Usalo prima del commit.
tools: Read, Grep, Glob
---

Sei un revisore attento. Cerca errori di
logica, casi limite e problemi di leggibilità.
```

> **Tip:** il comando `/agents` ti guida nella creazione di un sub-agent e può generarne una prima bozza da una descrizione.

Permessi granulari

I permessi base (`allow` / `deny`, cap. L2.4) decidono cosa Claude può fare senza chiedere. Gli hooks alzano il controllo: un `PreToolUse` può **valutare** la singola azione e decidere a runtime se permetterla, negarla o chiedere conferma. Così passi da regole statiche (“mai `rm -rf`”) a regole dinamiche (“blocca la scrittura fuori da `src/`”). Il principio resta quello del cap. L2.4: concedi il ripetitivo e sicuro, presidia il distruttivo.

In pratica: un hook che blocca

1. Apri `.claude/settings.json`.
2. Aggiungi un hook `PreToolUse` con il matcher del tool da sorvegliare.
3. Nel comando, controlla la condizione; se va bloccata, scrivi il motivo su stderr ed esci con **exit 2**.
4. Avvia Claude Code e prova un’azione che deve essere bloccata.
5. Verifica che Claude riceva il messaggio e si fermi.

Errori comuni

- **Hook che non blocca.** Manca l'**exit 2**: con altri codici l'azione prosegue.
- **Messaggio dove non serve.** Su un blocco, scrivi su **stderr**, non stdout: lo stdout viene ignorato.
- **Sub-agent senza description chiara.** Claude non sa quando delegargli il compito: cura la `description`.
- **Permessi troppo larghi.** Aprire tutto annulla la rete di sicurezza (cap. L2.4).

Riepilogo

1. Gli **hooks** eseguono tuoi comandi sugli eventi di Claude Code, da `settings.json` per matcher.
2. La **trappola dell'exit code**: solo **exit 2** blocca, e il motivo va su **stderr**.
3. I **sub-agent** delegano un compito a un contesto separato; Claude sceglie in base alla `description`.
4. I permessi granulari: gli hooks valutano a runtime cosa permettere.
5. Filosofia invariata: concedi il sicuro, presidia il distruttivo.

Prossimo passo

Nel **cap. L6.2 — MCP custom** vediamo come collegare a Claude ciò che non ha già un connettore pronto, e il pattern che unisce una skill a un server MCP.

Dati su hooks, sub-agent e permessi verificati il 24/06/2026 su code.claude.com/docs (hooks, sub-agents). Gli esempi di configurazione non sono stati eseguiti in questa sede.

Capitolo L6.2 — MCP custom

Livello 6 — Avanzato. Dati di prodotto verificati il 24/06/2026 su fonti ufficiali.

Obiettivo

Al termine capirai cos'è MCP in parole semplici, quando serve un server MCP custom invece di un connettore pronto, come aggiungerne uno a Claude Code, e il pattern che combina una skill con un MCP. È il modo per collegare Claude a ciò che non ha già un'integrazione.

Prerequisiti

- Aver usato i connettori (cap. L3.3).
- Claude Code installato (cap. L2.2), per la parte pratica.

Cos'è MCP, in parole semplici

MCP (Model Context Protocol) è uno **standard aperto** per collegare Claude a strumenti e dati esterni. Pensa a una presa elettrica universale: invece di un'integrazione su misura per ogni app, MCP definisce un modo comune perché Claude parli con un servizio. Un **server MCP** è il componente che espone a Claude le azioni di quel servizio.

I connettori del cap. L3.3 sono già MCP sotto il cofano: un **custom connector** è un **remote MCP** (un server raggiungibile via rete). MCP è dunque il livello tecnico che sta sotto ai connettori.

Quando serve un MCP custom

La directory dei connettori copre le app più diffuse. Un MCP custom serve quando:

- l'app che ti serve **non ha un connettore** pronto;

- vuoi collegare un tuo **strumento interno** o un database aziendale;
- ti serve un'azione specifica che nessun connettore esistente offre.

In breve: prima cerchi nella directory; se non c'è, costruisci o colleghi un MCP.

Locale o remoto

Un server MCP può girare in due modi, e Claude Code li gestisce entrambi.

Tabella L6.2.1 — MCP locale e remoto a confronto.

Tipo	Dove gira	Come
Locale (stdio)	sul tuo computer	child process
Remoto (HTTP)	su un server	via rete

Il **locale** (transport `stdio`) viene avviato da Claude Code come processo figlio, che comunica su standard input/output: utile per strumenti che vivono sulla tua macchina. Il **remoto** (transport `http`) è un server raggiungibile via rete: è la forma dei custom connector, raggiunti dal cloud di Anthropic (cap. L3.3).

Aggiungere un MCP a Claude Code

Dal terminale, il comando base è `claude mcp add`. Per un server locale:

```
claude mcp add --transport stdio \
  mio-tool -- npx -y mio-mcp-server
```

I flag (`--transport`, `--env`, `--scope`) vanno **prima** del nome; il `--` separa il nome dal comando che avvia il server. Le variabili d'ambiente si passano con `--env` (per esempio una API key del servizio).

> **Attenzione:** un MCP custom dà a Claude accesso a un servizio esterno. Collega solo server di cui ti fidi e non mettere segreti in chiaro: usa le variabili d'ambiente.

Il pattern skill + MCP

MCP e Skills risolvono problemi diversi e si completano. Un **MCP** dà a Claude la **capacità** di accedere a un servizio (i dati, le azioni). Una **skill** (Livello 5) dà a Claude il **metodo**: quando usare quella capacità, in che ordine, con quali regole.

Esempio: un MCP espone il tuo gestionale; una skill descrive il workflow «come preparare il report mensile dai dati del gestionale». L'MCP è il braccio, la skill è la procedura. Insieme rendono un'integrazione non solo possibile, ma **ripetibile**.

In pratica: collega uno strumento

1. Verifica prima nella directory dei connettori (cap. L3.3): se c'è, usa quello.
2. Se non c'è, procurati o scrivi un server MCP per il tuo strumento.
3. Aggiungilo con `claude mcp add`, scegliendo `stdio` (locale) o `http` (remoto).
4. Passa le credenziali con `--env`, mai in chiaro.
5. Se l'uso è ricorrente, scrivi una **skill** che ne descrive il workflow.

Errori comuni

- **Costruire un MCP che esiste già.** Controlla prima la directory dei connettori.
- **Flag dopo il nome.** Vanno prima; il `--` separa dal comando del server.

- **Segreti in chiaro.** Usa `--env`/variabili d'ambiente, non scriverli nel comando.
- **MCP senza skill per l'uso ripetuto.** L'MCP dà la capacità; la skill dà il metodo.

Riepilogo

1. **MCP** è lo standard aperto che collega Claude a strumenti e dati; un server MCP espone le azioni di un servizio.
2. I **custom connector** (cap. L3.3) sono **remote MCP**.
3. Serve un MCP custom quando non esiste un connettore pronto o per strumenti interni.
4. In Code: `claude mcp add` con transport `stdio` (locale) o `http` (remoto).
5. **Skill + MCP**: l'MCP dà la capacità, la skill il metodo ripetibile.

Prossimo passo

Nel **cap. L6.3 — Automazioni e controllo remoto** vediamo come far lavorare Claude da solo nel tempo: task pianificati, routine cloud e comando dal telefono.

Dati su MCP e `claude mcp add` verificati il 24/06/2026 su code.claude.com/docs/en/mcp e support.claude.com (custom connector). I comandi richiedono Claude Code e un server MCP, quindi non sono stati eseguiti in questa sede.

Capitolo L6.3 — Automazioni e controllo remoto

Livello 6 — Avanzato. Dati di prodotto verificati il 24/06/2026 su fonti ufficiali.

Obiettivo

Al termine saprai far lavorare Claude nel tempo senza starci davanti: i **Scheduled Tasks** in Cowork, le **Routines** cloud di Claude Code, e il **Dispatch** che trasforma il telefono in telecomando. Capirai soprattutto quale scegliere, perché girano in posti diversi.

Prerequisiti

- Cowork (cap. L3.1) per i task pianificati; Claude Code (cap. L2.2) per le routine.
- Un piano a pagamento.

Tre strumenti, tre posti

La differenza che conta non è cosa fanno, ma **dove girano**: dipende da questo se funzionano a computer spento, e quanto sono adatti al lavoro non tecnico.

Tabella L6.3.1 — Dove gira ciascuno strumento.

Strumento	Dove gira	A computer spento
Scheduled Task	Cowork, Desktop	no
Routine	cloud Anthropic	sì
Dispatch	il tuo computer	no

Scheduled Tasks

I **Scheduled Tasks** fanno girare un task di Cowork in automatico, su pianificazione o su richiesta: «ogni mattina controlla la posta», «ogni venerdì prepara il report». Vivono in **Cowork su Desktop**, sui piani a pagamento.

Il limite da conoscere: girano **solo a computer acceso e app Desktop aperta**. Se il computer è spento o l'app è chiusa all'ora prevista, il task viene **saltato** e parte da solo appena riaccendi o riapri l'app. Sono perfetti per chi lascia il computer acceso, meno per chi vuole garanzia di esecuzione a ogni costo.

Routines

Le **Routines** di Claude Code sono configurazioni salvate — un prompt, uno o più repository e i tuoi connettori — che girano sul **cloud di Anthropic**, non sul tuo computer. Per questo funzionano anche a macchina spenta.

Il loro punto di forza sono i **trigger** combinabili: la stessa routine può partire su pianificazione, in risposta a una chiamata API e a un evento GitHub, tutto insieme. Sono lo strumento per le automazioni di sviluppo che devono essere affidabili e indipendenti dalla tua macchina.

Dispatch

Il **Dispatch** trasforma il telefono in un telecomando per Claude sul tuo computer. Mandi un messaggio dall'app mobile e Claude esegue il task **sulla tua macchina**: legge i file locali, tira dati dai connettori, cerca sul web e ti riporta il risultato. Tra l'app mobile e il Desktop si apre un **thread unico e persistente**, come un walkie-talkie: tu assegni da remoto, Claude lavora in locale.

È la risposta al «vorrei far partire quel lavoro mentre sono fuori»: il computer deve essere acceso, ma tu no.

Quale scegliere

- Lavoro **non tecnico, ricorrente**, e tieni il computer acceso → **Scheduled Task**.
- Automazione di **sviluppo** che deve girare **sempre**, anche a macchina spenta, con trigger vari → **Routine** cloud.
- Vuoi **avviare da remoto** un lavoro che usa i tuoi file locali → **Dispatch**.

In pratica: pianifica un task ricorrente

1. In Cowork, descrivi il task come faresti per un lavoro normale (cap. L3.1).
2. Imposta la ricorrenza (per esempio «ogni mattina alle 8»).
3. Lascia il computer acceso e l'app Desktop aperta all'orario previsto.
4. Per le automazioni di sviluppo indipendenti, valuta invece una **Routine** in Claude Code.
5. Per avviare da fuori, usa il **Dispatch** dall'app mobile.

Errori comuni

- **Scheduled Task a computer spento.** Viene saltato e parte alla riapertura: per l'esecuzione garantita usa una Routine cloud.
- **Confondere Routine e Scheduled Task.** Le Routine girano sul cloud (Code), i task in Cowork sul tuo Desktop.
- **Aspettarsi il Dispatch a macchina spenta.** Esegue in locale: il computer deve essere acceso.

Riepilogo

1. La differenza chiave è **dove gira** lo strumento.
2. **Scheduled Tasks**: Cowork su Desktop; solo a computer acceso e app aperta.
3. **Routines**: cloud di Anthropic (Code); girano sempre, trigger combinabili.
4. **Dispatch**: avvii dal telefono, Claude esegue sul tuo computer acceso.
5. Scegli in base ad affidabilità richiesta e tipo di lavoro.

Prossimo passo

Nel **cap. L6.4 — Gestire i limiti d’uso** vediamo da cosa dipende il consumo e come lavorare a lungo senza sbattere contro i limiti.

Dati su Scheduled Tasks, Routines e Dispatch verificati il 24/06/2026 su support.claude.com (scheduled tasks, dispatch) e code.claude.com/docs (scheduled-tasks/routines). Le funzioni richiedono un account a pagamento, quindi non sono state eseguite in questa sede.

Capitolo L6.4 — Gestire i limiti d’uso

Livello 6 — Avanzato. Dati di prodotto verificati il 02/07/2026 su fonti ufficiali.

Obiettivo

Al termine capirai la differenza tra **usage limit** e **length limit**, da cosa dipende il consumo, e come lavorare a lungo senza sbattere contro i limiti. Sono le nozioni che ti fanno usare Claude in modo sostenibile, soprattutto su task pesanti.

Prerequisiti

- Conoscere i piani (cap. F.3) e i Project (cap. L3.2).

Due limiti diversi

Claude ha due tipi di limite, che funzionano in modo distinto:

- **Usage limit:** *quanto* puoi usare Claude in un periodo. È il tuo “budget di conversazione”. Al suo esaurirsi, aspetti il reset.
- **Length limit:** *quanto* può diventare lunga una singola chat, cioè il **context window** (la memoria di lavoro di Claude in quella conversazione).

Il primo riguarda la quantità nel tempo; il secondo la profondità di una singola chat. Confonderli porta alle scelte sbagliate quando qualcosa si “blocca”.

Da cosa dipende il consumo

L'usage non si consuma a colpi fissi. Pesano più fattori: la **lunghezza e complessità** della conversazione, le **feature** attive, il **modello** scelto e l'**effort level** (quanto a fondo ragiona). Un dettaglio che sorprende: tutte le superfici — claude.ai, Claude Code, Claude Desktop — attingono allo **stesso** usage. Non sono budget separati.

> **Nota:** con la Code execution attiva, le chat lunghe innescano la **gestione automatica del contesto**: Claude riassume i messaggi più vecchi per proseguire. Comodo, ma le conversazioni lunghe consumano **più** usage.

Il context window

Nella chat, sui piani a pagamento, il context window arriva a **1M token** con Sonnet 5 (il primo modello a offrirlo in chat), è di **500K** per gli altri modelli di punta (Opus 4.6/4.7/4.8 e Sonnet 4.6) e di **200K** per il resto. In **Claude Code** i modelli di punta arrivano fino a **1M token**: su Pro, per il milione con Opus, vanno abilitati gli usage credits. Non puoi ingrandirlo a piacere, ma puoi usarlo meglio: è lì che si gioca la differenza tra una chat che regge e una che si satura.

Task economici: ridurre il consumo

Alcune mosse abbassano il consumo a parità di risultato:

- **Usa i Project.** Sfruttano il RAG (recupero mirato): caricano in contesto solo ciò che serve, invece di tutto.
- **Istruzioni brevi.** Project e prompt concisi pesano meno e rendono di più.
- **Togli ciò che non serve.** File inutili nei Project, e **tool/connettori** non necessari: sono token-intensive, spegnili quando non servono.

- **Abbassa l’effort** e disattiva l’**extended thinking** sui task di routine.
- **Scegli il modello giusto.** Non usare il più potente “per sicurezza” (cap. F.3): spesso paghi di più senza vantaggi.

Cosa fare al limite

Quando sbatti contro un limite, la mossa dipende da quale:

- **Usage limit raggiunto:** aspetti il **reset**, fai **upgrade** di piano, o compri **usage credits** (sui piani a pagamento) per continuare subito.
- **Length limit (chat troppo lunga):** apri una **nuova conversazione**, o sposta il materiale in un **Project** per lavorarci in modo più efficiente.

> **Nota:** i crediti d’uso non servono solo “al limite”: alcuni modelli di punta si usano *soprattutto* così. Al rientro di Fable 5 (luglio 2026), per esempio, il modello era incluso nei piani a pagamento solo in parte e per pochi giorni; passato il periodo, l’accesso è rimasto solo via usage credits.

> **Tip:** se sei in una chat lunga e vicino al limite d’uso, conviene spesso ricominciare in una chat nuova: riparti con il contesto essenziale invece di trascinarti dietro tutto lo storico.

In pratica: lavora a lungo senza bloccarti

1. Per un lavoro ricorrente, mettilo in un **Project** invece di chat infinite.
2. Tieni istruzioni brevi e rimuovi i file che non usi più.
3. Disattiva tool e connettori non necessari per quella sessione.
4. Sui task semplici, abbassa l’effort e spegni l’extended thinking.
5. Se ti avvicini al limite in una chat lunga, **aprine una nuova**.

Errori comuni

- **Confondere i due limiti.** Se è la chat a essere piena, non serve aspettare il reset: apri una chat nuova.
- **Tenere tutto acceso.** Tool e connettori inutili consumano contesto e usage: spegnili.
- **Modello sovradimensionato.** Più potenza del necessario costa di più senza resa migliore.
- **Chat infinite.** Trascinare uno storico enorme consuma usage: spezza in chat o usa i Project.

Riepilogo

1. **Usage limit** = quanto usi nel tempo; **length limit** = quanto è lunga una chat (context window).
2. Il consumo dipende da lunghezza, feature, **modello** ed **effort**; tutte le superfici contano sullo **stesso** usage.
3. Context window in chat: fino a **1M** (Sonnet 5), **500K** per gli altri modelli di punta, **200K** per il resto; in Claude Code fino a **1M** (con usage credits su Pro per Opus).
4. Riduci il consumo con Project, istruzioni brevi, meno tool attivi, effort più basso.
5. Al limite: reset/upgrade/**credits** per l’usage; **nuova chat**/Project per la lunghezza.

Prossimo passo

Nel **cap. L6.5 — Claude al lavoro, sicuro** vediamo come introdurre Claude in un’organizzazione: governance, ruoli, dati e quando non usare l’account personale.

Dati su usage/length limit e context window verificati il 02/07/2026 su support.claude.com/en/articles/11647753 e /articles/8606394. Valori soggetti a modifica: vedi il ledger e le fonti ufficiali per gli aggiornamenti.

Capitolo L6.5 — Claude al lavoro, sicuro

Livello 6 — Avanzato. Dati di prodotto verificati il 24/06/2026 su fonti ufficiali.

Obiettivo

Al termine saprai cosa cambia quando Claude entra in un'organizzazione: chi governa le capacità, come si gestiscono ruoli e dati, e perché per il lavoro aziendale l'account personale non è la scelta giusta. È il capitolo che separa l'uso individuale da quello di squadra.

Prerequisiti

- Conoscere i piani (cap. F.3) e i connettori (cap. L3.3).

Account personale o aziendale

La prima decisione è anche la più importante: per il lavoro dell'organizzazione si usa un account **gestito dall'organizzazione** (Team o Enterprise), non quello personale. La ragione non è formale. Su un account aziendale i dati, gli accessi e le regole li controlla chi amministra; su quello personale restano in mano al singolo, fuori dalle policy dell'azienda. Mettere materiale di lavoro su un account personale significa portarlo fuori dal perimetro che l'organizzazione può proteggere e gestire.

Chi governa le capacità

In Team ed Enterprise non tutto è acceso per tutti. Un **Owner** o **Primary Owner** decide cosa è disponibile, da **Admin > Capabilities**: Web search, connettori, e altre funzioni si abilitano a livello di organizzazio-

ne. Per questo, se non vedi una funzione, spesso non è un errore: è una scelta di chi amministra (lo abbiamo già visto per Web search nel cap. L1.3 e per i connettori nel cap. L3.3).

Lo stesso vale per le **azioni dei connettori**: un admin può permettere la sola lettura e bloccare la scrittura, per tutta l'organizzazione (cap. L3.3).

Ruoli, identità e dati

Le organizzazioni hanno bisogno di controlli che un account singolo non ha. La tabella riassume cosa aggiungono i piani per le aziende.

Tabella L6.5.1 — Controlli per le organizzazioni.

Ogni colonna è additiva: Enterprise include quel che ha Team, e aggiunge.

Area	Team	Enterprise (in più)
Accessi	admin, SSO	RBAC, SCIM
Dati	no training*	retention, audit log
Rete	—	IP allowlisting

*Sia su Team sia su Enterprise, di default i dati non sono usati per il training; Enterprise aggiunge retention e audit log.

In sintesi: **Team** porta amministrazione, SSO (accesso unico) e deployment controllato; **Enterprise** aggiunge ruoli granulari (RBAC), provisioning automatico degli utenti (SCIM), audit log, gestione della retention dei dati, allowlisting degli IP e funzioni di sicurezza dedicate. Sono gli strumenti che rendono l'uso conforme alle policy aziendali, non lasciato al singolo.

Il modello dei permessi, ancora

Un principio attraversa tutto il libro: Claude **eredita i tuoi permessi**. Vale per i connettori (vede solo ciò che vedi tu alla fonte, cap. L3.3) e per il lavoro sui file (tocca solo le cartelle collegate, cap. L3.1). In azienda questo è una garanzia: i controlli del sistema d'origine restano in vigore, e Claude non li scavalca. Restringere in Claude non concede mai più accesso di quanto la fonte permetta.

In pratica: introdurre Claude in azienda

1. Usa un account **gestito dall'organizzazione**, non quello personale.
2. Fai definire a un Owner le **capacità** disponibili (Admin > Capabilities).
3. Imposta gli **accessi**: SSO e, in Enterprise, ruoli (RBAC) e provisioning (SCIM).
4. Configura le **restrizioni dei connettori** secondo cosa il team può leggere o modificare (cap. L3.3).
5. Concorda le regole su **dati** e retention prima di mettere materiale sensibile.

Errori comuni

- **Lavoro aziendale su account personale.** Porta i dati fuori dal controllo dell'organizzazione: usa l'account gestito.
- **Pensare a un bug quando manca una funzione.** Spesso l'ha disattivata un admin (cap. L1.3, L3.3).
- **Connettori senza restrizioni.** In team, definisci cosa è in sola lettura e cosa no (cap. L3.3).
- **Rinviare le regole sui dati.** Concordale prima, non dopo aver caricato materiale sensibile.

Riepilogo

1. Per il lavoro aziendale usa un account **gestito dall'organizzazione**, non personale.
2. Un **Owner** governa le capacità da **Admin > Capabilities**.
3. **Team** aggiunge admin e SSO; **Enterprise** RBAC, SCIM, audit log, retention, IP allowlisting.
4. Claude **eredita i permessi**: i controlli della fonte restano in vigore.
5. Definisci accessi, restrizioni dei connettori e regole sui dati prima di partire.

Prossimo passo

Nel **cap. L6.6 — Integrare via API** chiudiamo il livello tecnico: come passare dalle interfacce al codice, con l'endpoint dei messaggi e un primo esempio.

Dati su controlli Team/Enterprise (RBAC, SCIM, SSO, audit, retention) verificati il 24/06/2026 su claude.com/pricing e support.claude.com. Pubblicazione indipendente, non affiliata ad Anthropic.

Capitolo L6.6 — Integrare via API

Livello 6 — Avanzato. Dati di prodotto verificati il 24/06/2026 su fonti ufficiali.

Obiettivo

Al termine saprai quando passare alle API, com'è fatta una chiamata all'endpoint dei messaggi, e come funziona il **tool use** (l'uso degli strumenti). È il ponte da chi *usa* Claude a chi lo *integra* nei propri software.

Prerequisiti

- Una API key dalla Console (cap. L2.3, F.3).
- Dimestichezza con un linguaggio di programmazione.

Quando servono le API

Le interfacce (chat, Cowork, Code) bastano per il lavoro personale. Le **API** servono quando vuoi mettere Claude **dentro** un tuo prodotto: un'app che risponde ai clienti, un processo che classifica documenti, un servizio che genera testo su richiesta. È accesso programmatico, pay-as-you-go a token (cap. F.3).

L'endpoint dei messaggi

Il cuore delle API è un endpoint: invii una lista di messaggi, Claude genera il successivo. La chiamata grezza, con `curl`:

```
curl https://api.anthropic.com/v1/messages \
  -H "x-api-key: $ANTHROPIC_API_KEY" \
  -H "anthropic-version: 2023-06-01" \
  -H "content-type: application/json" \
  -d '{
    "model": "claude-opus-4-8",
    "max_tokens": 1024,
    "messages": [
      {"role": "user",
       "content": "Ciao"}
    ]
  }'
```

Tre header sono obbligatori: `x-api-key` (la tua chiave), `anthropic-version` (es. `2023-06-01`) e `content-type`. Il corpo dichiara almeno `model`, `max_tokens` e `messages`. (VOLATILE: la stringa del modello cambia, vedi il ledger.)

Con l'SDK

In pratica non si usa `curl`, ma un SDK ufficiale (Python o TypeScript), che gestisce header e formati. In Python:

```
import anthropic

# Reads the key from ANTHROPIC_API_KEY
client = anthropic.Anthropic()

msg = client.messages.create(
    model="claude-opus-4-8",
    max_tokens=1024,
    messages=[
        {"role": "user",
         "content": "Ciao, chi sei?"},
    ],
)
print(msg.content[0].text)
```

Tool use, in breve

Il **tool use** permette a Claude di usare strumenti che fornisci tu — una funzione che interroga un database, chiama un'API, fa un calcolo. Il flusso è un botta e risposta in tre tempi:

1. Definisci i tool nella richiesta e mandi il messaggio.
2. Se serve uno strumento, Claude risponde con `stop_reason: "tool_use"` e un blocco che dice **quale** tool e con **quali** argomenti.
3. Il tuo codice esegue l'operazione e rimanda il risultato come `tool_result`; Claude lo usa per la risposta finale.

Figura L6.6.1 — Il ciclo del tool use. Alt-text: diagramma verticale dal messaggio alla richiesta di tool, all'esecuzione nel tuo codice, al risultato, alla risposta finale.

Il punto chiave: lo strumento **lo esegui tu**, nel tuo software. Claude decide quando usarlo e con quali argomenti, ma il codice resta sotto il tuo controllo.

In pratica: la tua prima chiamata

1. Genera una **API key** nella Console e impostala come variabile d'ambiente (cap. L2.3).
2. Installa l'SDK del tuo linguaggio (Python o TypeScript).
3. Fai una chiamata minima all'endpoint dei messaggi e leggi la risposta.
4. Per integrare azioni, definisci un **tool** e gestisci il ciclo `tool_use` → `tool_result`.
5. Tieni la chiave fuori dal codice: usa variabili d'ambiente (cap. L2.3).

Errori comuni

- **Header mancanti.** Servono `x-api-key`, `anthropic-version` e `content-type`.
- **Stringa del modello fissata.** Cambia nel tempo: prendila dal ledger o dalle fonti ufficiali.
- **Chiave nel codice.** Mai: usa una variabile d'ambiente (cap. L2.3).
- **Aspettarsi che Claude esegua il tool.** Lo esegui tu; Claude decide quando e come chiamarlo.

Riepilogo

1. Le API servono a integrare Claude **dentro** i tuoi software; pay-as-you-go.
2. Endpoint `POST /v1/messages` con header `x-api-key`, `anthropic-version`, `content-type`.
3. In pratica usi un **SDK** (Python/TS), non `curl`.

4. Il **tool use** è un ciclo: Claude chiede un tool, il tuo codice lo esegue e rimanda il risultato.
5. La chiave sta in una **variabile d’ambiente**, mai nel codice.

Prossimo passo

Hai completato il Livello 6. Nella **Chiusura**, il **cap. C.1 — Progetto end-to-end** attraversa tutti i livelli su un caso reale, dal design al codice fino all’automazione.

Dati su endpoint, header e tool use verificati il 24/06/2026 su platform.claude.com/docs (messages, tool use). Gli esempi non sono stati eseguiti in questa sede; richiedono una API key valida.

Capitolo C.1 — Progetto end-to-end

Chiusura — livello variabile. Concetti di sintesi; dettagli di prodotto dal ledger (verificati 24/06/2026).

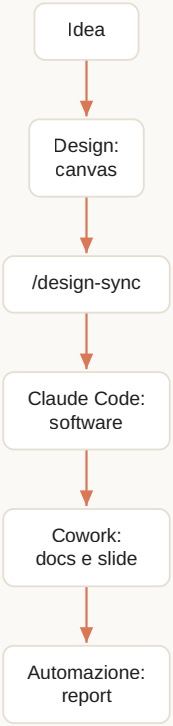
Obiettivo

Questo capitolo attraversa tutti i livelli del libro su un caso reale: da un'idea a un piccolo prodotto, passando per Design, Claude Code, Cowork e l'automazione. Non introduce nulla di nuovo: mostra i pezzi già visti che lavorano **insieme**.

Il caso

Immagina di voler costruire una piccola app interna: una pagina dove il team registra le richieste dei clienti e ne segue lo stato. La porteremo dall'idea fino a un report che si aggiorna da solo. Ogni tappa rimanda al capitolo dove l'hai imparata.

Figura C.1.1 — Il flusso completo, da idea a automazione. Alt-text: diagramma verticale che collega idea, Design, /design-sync, Code, Cowork e automazione in sequenza.



1. Idea e fondamenta

Si parte dalla chat (Livello 1). Descrivi l'idea e fatti aiutare a metterla a fuoco: cosa deve fare la pagina, per chi, con quali campi. Vale la regola del cap. L1.2 — scopo, destinatario, formato — e quella del cap. L1.1: la prima risposta è una bozza da affinare. Da qui esce un breve documento di requisiti.

2. Dal design al codice

Apri **Claude Design** (cap. L4.1) e generi la pagina sul canvas, partendo dal tuo design system se l'hai importato (cap. L4.2). Rifinisci con chat, commenti e modifiche dirette. Quando l'aspetto regge, usi `/design-`

`sync` e l'**handoff** verso **Claude Code** (cap. L4.3): il design diventa software vero, senza ricostruirlo da uno screenshot.

In Claude Code lavori sul progetto reale, con `CLAUDE.md` e i permessi configurati (cap. L2.4). Per i passaggi delicati, un **hook** impone i controlli e un **sub-agent** rivede il codice prima del commit (cap. L6.1). Se l'app deve leggere da un tuo strumento interno, lo colleghi via **MCP** (cap. L6.2).

3. Documenti e organizzazione

A app avviata, serve materiale di contorno: una guida d'uso, una presentazione per il team. Qui entra **Cowork** (cap. L3.1): colleghi la cartella del progetto e gli affidi il lavoro come **end-state** — «da questi file, genera una guida `.docx` e una presentazione `.pptx`». Valgono i metodi del Livello 3: prima il contenuto, poi il file (cap. L3.4), e struttura → contenuti → stile per le slide (cap. L3.5).

Una **skill** di progetto (Livello 5) tiene insieme le regole — voce, struttura, nomi dei file — così l'output è coerente ogni volta (cap. L5.3).

4. Automazione

Infine, il report che si aggiorna da solo. Con un **Scheduled Task** in Cowork (cap. L6.3) fai sì che ogni venerdì Claude rilegga le richieste e produca un riepilogo. Se ti serve indipendente dalla tua macchina, usi una **Routine** cloud; se vuoi farlo partire da fuori, il **Dispatch** dal telefono. E se l'app cresce fino a richiedere un'integrazione vera, passi alle **API** (cap. L6.6).

Lungo tutto il percorso tieni d'occhio i **limiti d'uso** (cap. L6.4): Project per il contesto, sessioni pulite, tool accesi solo quando servono.

Cosa dimostra

Il valore non sta nel singolo strumento, ma nel passaggio di consegne: Design passa a Code, Code lavora con Cowork sulla stessa cartella, una skill rende tutto ripetibile, l'automazione chiude il cerchio. È la differenza, vista fin dal cap. F.2, tra una somma di strumenti e un ecosistema.

In pratica: prova un mini-progetto

1. In chat, metti a fuoco un'idea piccola e concreta (cap. L1.2).
2. Genera l'interfaccia in Design e fai handoff a Code (cap. L4.3).
3. Configura il progetto in Claude Code (cap. L2.4).
4. In Cowork, genera guida e slide dalla cartella (cap. L3.4, L3.5).
5. Aggiungi un'automazione per il report ricorrente (cap. L6.3).

Riepilogo

1. Un progetto reale attraversa tutti i livelli: chat, Design, Code, Cowork, automazione.
2. **Design** → **/design-sync** → **Code** porta l'interfaccia a software senza ricostruirla.
3. **Cowork** produce documenti e slide dalla stessa cartella del progetto.
4. Una **skill** rende l'output coerente; l'**automazione** lo mantiene aggiornato.
5. Il valore è nel passaggio di consegne tra i prodotti, non nel singolo strumento.

Prossimo passo

Nel **cap. C.2 — Appendici** trovi il glossario, una tabella di troubleshooting e i link ufficiali, da consultare quando servono.

Capitolo di sintesi: richiama i prodotti trattati nei Livelli 1-6. Nessun comando eseguito in questa sede; i rimandi puntano ai capitoli dove ogni passo è spiegato.

Capitolo C.2 — Appendici

Chiusura — materiale di consultazione. Dati di prodotto verificati il 24/06/2026 su fonti ufficiali.

Come usare queste appendici

Non si leggono in sequenza: si consultano. Trovi un glossario dei termini ricorrenti, una tabella di troubleshooting per i problemi più comuni, e i link ufficiali da cui verificare i dati volatili.

Glossario

- **Agentico:** modo di lavorare in cui Claude porta avanti un compito in più passi, non solo una risposta singola.
- **API:** accesso programmatico ai modelli, per integrare Claude nei propri software (cap. L6.6).
- **Canvas:** la tela di Claude Design dove appaiono le interfacce generate (cap. L4.1).
- **Connettore (Connector):** collegamento che dà a Claude accesso a un'app, con i tuoi permessi (cap. L3.3).
- **Context window:** la memoria di lavoro di una singola chat, misurata in token (cap. L6.4).
- **Cowork:** lavoro agentico non di codice nell'app desktop, in VM isolata (cap. L3.1).
- **Design /** `/design-sync`**:** il canvas e il comando che sincronizza design e codice nei due versi (cap. L4.3).
- **Dispatch:** avviare da telefono un task che Claude esegue sul tuo computer (cap. L6.3).
- **Effort level:** quanto a fondo Claude ragiona; più alto consuma più usage (cap. L6.4).

- **Handoff:** passaggio di un design a Claude Code perché diventi software (cap. L4.3).
- **Hook:** comando agganciato a un evento di Claude Code (cap. L6.1).
- **MCP:** standard aperto che collega Claude a strumenti e dati esterni (cap. L6.2).
- **Native installer:** metodo consigliato per installare Claude Code, senza Node (cap. L2.2).
- **OAuth:** accesso delegato via browser, senza digitare la password nel terminale (cap. L2.3).
- **Prompt:** la richiesta che scrivi a Claude (cap. L1.2).
- **Project:** workspace con istruzioni e knowledge base condivise tra le chat (cap. L3.2).
- **Routine:** automazione di Claude Code che gira sul cloud (cap. L6.3).
- **Scheduled Task:** task ricorrente di Cowork, gira a computer acceso (cap. L6.3).
- **Skill /** `SKILL.md` **:** conoscenza riutilizzabile che Claude carica da solo (cap. L5.1).
- **Sub-agent:** assistente con contesto e permessi propri, a cui Code delega (cap. L6.1).
- **Tool use:** uso, via API, di strumenti che fornisci tu (cap. L6.6).
- **Usage limit:** quanto puoi usare Claude in un periodo (cap. L6.4).
- **VM isolata:** macchina virtuale in cui gira Cowork; vede solo le cartelle collegate (cap. L3.1).

Troubleshooting

Tabella C.2.1 — Problemi comuni e prima soluzione.

Problema	Causa	Soluzione
`command not found`	PATH	nuovo terminale (L2.2)
Login fallito	API key di troppo	`unset` la key (L2.3)
Funzione assente	admin	Admin > Capabilities (L1.3)
Hook non blocca	exit code	usa exit 2 (L6.1)
Connettore inerte	non attivato	toggle in chat (L3.3)
Task saltato	PC spento	riapri l'app (L6.3)
Chat satura	length limit	nuova chat (L6.4)

Link ufficiali

Da qui verifichi i dati che cambiano nel tempo. Sono le uniche fonti su cui questo libro si basa per i dettagli di prodotto.

- **claude.ai** — accesso a Claude (web).
- **claude.com** e **claude.com/pricing** — prodotti e piani.
- **support.claude.com** — guide e troubleshooting ufficiali.
- **code.claude.com/docs** — documentazione di Claude Code.
- **platform.claude.com/docs** — documentazione API e Console.
- **agentskills.io** — standard aperto delle Skills.
- **github.com/anthropics/skills** — skill di esempio.

Note legali

Pubblicazione **indipendente, non affiliata né approvata da Anthropic**. Claude e Anthropic sono marchi dei rispettivi titolari. I dati di prodotto (versioni, comandi, prezzi, menu, piani) sono **verificati alla data indicata** in ciascun capitolo e nel ledger, e sono **soggetti a modifica**: per i valori correnti fai riferimento alle fonti ufficiali sopra e al repo companion.

Riepilogo

1. Il **glossario** raccoglie i termini ricorrenti, con il rimando al capitolo.
2. La tabella di **troubleshooting** dà la prima mossa per i problemi comuni.
3. I **link ufficiali** sono le sole fonti per i dati volatili.
4. La pubblicazione è **indipendente** e non affiliata ad Anthropic.
5. I dati sono verificati alla data indicata e vanno ricontrollati alle fonti.

Fine

Hai attraversato tutto il percorso: dai primi messaggi in chat fino alle API e all'automazione. Il prodotto continuerà a cambiare — per questo i dati volatili sono datati e raccolti nel repo companion. Le idee di fondo, invece, restano: descrivi il risultato, dai contesto, itera, e fai collaborare i pezzi dell'ecosistema.

Materiale di consultazione. Glossario e rimandi sono evergreen; troubleshooting, link e dati di prodotto verificati il 24/06/2026 sulle fonti ufficiali elencate.