Guida pratica 10 min di setup

Installa il sistema di memoria sui tuoi progetti AI

Guida passo-passo per partire da zero. Nessuna configurazione manuale: copi un prompt, Claude installa tutto da solo sulla tua cartella.

Come funziona

Incolli un prompt in Claude Code con la tua cartella progetto aperta. Claude analizza il tuo progetto e crea automaticamente il file di memoria, lo script di aggiornamento e la struttura a 3 strati. Lanci un comando e sei operativo.

Cosa ti serve

Servono solo due cose, entrambe gratuite. Se ce le hai gia', salta al metodo rapido.

1. Claude Code (app desktop)

E' l'app desktop di Claude, diversa dal sito web. Permette a Claude di leggere e scrivere file sul tuo computer.

Scarica da: **claude.ai/download**

2. Python (linguaggio di programmazione)

Non devi saper programmare. Ti serve solo per far girare lo script automatico di aggiornamento.

Scarica da: **python.org/downloads**. Installa con le opzioni di default.

Verifica: apri il Terminale (Mac) o Prompt dei comandi (Windows), scrivi `python --version` e premi Invio. Se vedi un numero tipo *Python 3.11.2*, sei a posto.

Il metodo rapido

Tre passi. Claude fa il lavoro per te.

PASSO 1

Apri la cartella progetto in Claude Code

Dal menu: **File → Open Folder**. Seleziona la cartella dove vuoi installare il sistema di memoria.

Se parti da zero senza cartella, creane una vuota sul Desktop e aprila.

PASSO 2

Copia e incolla questo prompt nella chat

Copia **tutto** il contenuto della box sotto e incollalo nella chat di Claude. Non modificare nulla.

PROMPT DA COPIARE

```
Installa il sistema di memoria a 3 strati in questo progetto.

Prima analizza la cartella corrente (fai ls ricorsivo), capisci il tipo di progetto
e lo stack. Se la cartella e' vuota chiedimi: nome progetto, descrizione in una frase,
cartelle che prevedo di creare.

Poi crea questi file nella root del progetto:

=== FILE 1: memoria.py ===
Script Python con CONFIG parametrica in testa e questi comandi:
- start: scansiona SCAN_DIRS, confronta hash file con .snapshot.json,
  aggiorna sezione AUTO:DIFF in memoria/CONTEXT.md, salva nuovo snapshot,
  stampa riepilogo cambiamenti
- end "desc": appende riga a storico/sessioni.md, pulisce diff in CONTEXT.md,
  salva snapshot
- sync: rigenera sezioni AUTO senza cambiare sessione
- check: verifica righe CONTEXT.md vs limite, staleness dettaglio/
- stats: statistiche memoria

CONFIG da adattare al progetto rilevato:
  SCAN_DIRS = [cartelle trovate o ["src"] se vuoto]
  SCAN_EXTENSIONS = {estensioni rilevanti per il linguaggio trovato}
  IGNORE_DIRS = {"venv",".venv","node_modules","__pycache__",".git","dist","build"}
  IGNORE_FILES = {".env","database.db",".snapshot.json"}
  CONTEXT_MAX_LINES = 150
```

```
Usa solo libreria standard Python (no pip). No emoji. Encoding utf-8 con fallback cp1252.

═══ FILE 2: memoria/CONTEXT.md ═══
Compila con le info reali del progetto rilevato:

# CONTEXT - {{nome progetto}}
> Trascina SOLO questo file a inizio sessione. Scrivi START per partire.
> Ultimo sync: {{data oggi}}

---

## ISTRUZIONI CLAUDE (non cancellare)

**Quando l'utente scrive START:**
1. Esegui: python memoria.py start
2. Leggi l'output. Se ci sono file cambiati (~) o nuovi (+), apri SOLO quelli
3. Rispondi: "Allineato. [riepilogo diff]. Cosa facciamo?"

**Quando l'utente scrive STOP** (Claude fa tutto, senza chiedere):
1. Sintetizza la sessione in 1 riga
2. Aggiorna BUG (rimuovi fixati), TODO (rimuovi fatti), REGOLE (se nuove)
3. Aggiorna file in memoria/dettaglio/ se ci sono cambiamenti rilevanti
4. Esegui: python memoria.py end "descrizione auto-generata"
5. Rispondi: riepilogo + "Prossima volta trascina CONTEXT.md"

**Path progetto:** {{path assoluto}}

---

## PROGETTO
{{descrizione 1-2 righe}}
Stack: {{stack rilevato}}

---

←!— AUTO:DIFF —→
Nessuna modifica dall'ultima sessione.
←!— /AUTO:DIFF —→

---

## BUG APERTI [MANUALE]
(nessuno ancora)

## TODO ATTIVI [MANUALE]
- [ ] Primo sync: lanciare python memoria.py start
```

```
## REGOLE CRITICHE [MANUALE]
(aggiungi le convenzioni da non dimenticare mai)

---

## MAPPA DETTAGLIO
| Serve info su...    | Leggi                              |
|---------------------|------------------------------------|
| Stack, struttura    | memoria/dettaglio/architettura.md |
| Server, config, env | memoria/dettaglio/config.md       |
| Decisioni passate   | memoria/dettaglio/decisioni.md    |
| Storico sessioni    | memoria/storico/sessioni.md       |

═══ FILE 3: memoria/dettaglio/architettura.md ═══
Header con last_verified: {{oggi}}, sezione Stack pre-compilata con lo stack rilevato,
sezioni vuote: Struttura cartelle, Endpoint/Entry point, Dipendenze esterne.

═══ FILE 4: memoria/dettaglio/config.md ═══
Header con last_verified: {{oggi}}, sezioni vuote con commenti guida:
Server produzione, Domini e SSL, Variabili d'ambiente, Deploy, Note operative.

═══ FILE 5: memoria/dettaglio/decisioni.md ═══
Header con last_verified: {{oggi}} + istruzione append-only.
Formato entry: ## YYYY-MM-DD | Titolo / Cosa / Perche' / Alternative scartate.

═══ FILE 6: memoria/storico/sessioni.md ═══
Solo header tabella: | # | Data | Descrizione | Diff |

═══ FILE 7: memoria/storico/archivio.md ═══
Solo header: # Archivio TODO Completati

Al termine dimmi: file creati, SCAN_DIRS impostato e perche', poi scrivi
"Ora lancia: python memoria.py start dal terminale nella cartella del progetto"
```

PASSO 3

Primo sync dal terminale

Quando Claude ha finito, apri il Terminale (Mac) o Prompt dei comandi (Windows), entra nella cartella del tuo progetto e lancia:

```
python memoria.py start
```

Questo crea la prima "foto" del tuo progetto. Da ora il sistema e' attivo.

Come lavori ogni giorno

Dopo il setup iniziale, ogni sessione segue questo ciclo. Due comandi: **START** e **STOP**.

Inizio sessione

1. Dal terminale: `python memoria.py start`
2. Trascina `memoria/CONTEXT.md` nella chat di Claude
3. Scrivi `START`

Claude legge il contesto, confronta con l'ultima foto, capisce cosa e' cambiato. Poi lavorate.

Fine sessione

1. Scrivi `STOP` nella chat

Claude aggiorna da solo bug, TODO, decisioni. Lancia lo script di chiusura. Tu non tocchi niente.

Cosa viene creato nella tua cartella

```
memoria.py ← script che aggiorna tutto
memoria/
  CONTEXT.md ← il file che trascini in chat ogni volta
  dettaglio/
    architettura.md ← stack, struttura
    config.md ← server, variabili
    decisioni.md ← log scelte fatte
  storico/
    sessioni.md ← tutte le sessioni
    archivio.md ← TODO completati
```

Quello che usi ogni giorno

Solo **memoria/CONTEXT.md**. Il resto Claude lo apre quando serve. Tu non devi preoccuparti.

Comandi utili

Lo script ha cinque comandi, ma nel 95% dei casi ti serve solo `start`. Gli altri sono opzionali.

`python memoria.py start` — inizio sessione (obbligatorio)

`python memoria.py end "descrizione"` — fine sessione (lo fa Claude con STOP)

`python memoria.py sync` — rigenera CONTEXT.md se qualcosa sembra strano

`python memoria.py check` — health check del sistema

`python memoria.py stats` — statistiche sulle sessioni

Domande frequenti

Devo trascinare CONTEXT.md ogni volta?

Si'. E' il punto: trascini un solo file invece di rispiegare tutto da capo. Di solito sono 100–150 righe, in un clic.

Posso usarlo su piu' progetti?

Si'. Ogni progetto ha la sua cartella `memoria/` indipendente. Installi il sistema una volta per progetto, funziona su tutti in parallelo.

E se dimentico di scrivere STOP?

Il progetto non si rompe. La prossima sessione `python memoria.py start` mostra comunque le differenze rilevate. Perdi solo il riassunto testuale di quella sessione.

Lo script modifica i miei file di progetto?

No. Lo script li legge per confrontarli tra loro, ma non li tocca mai. Scrive solo dentro la cartella `memoria/`.

Posso modificare CONTEXT.md a mano?

Si'. Le sezioni manuali (BUG, TODO, REGOLE) le gestisci tu e Claude. Le sezioni AUTO vengono rigenerate dallo script ogni `start`.

Funziona anche su progetti non tecnici?

Si'. Qualsiasi cartella con file testuali: documenti, testi, analisi, ricerche. Lo script traccia qualsiasi tipo di file.

Un esempio concreto

Hai un sito con 30 file. Stai costruendo una pagina contatti.

Senza sistema, ogni sessione ricominci da capo:

> Ciao Claude, sto lavorando su un sito, ho 30 pagine, la struttura e' questa, i colori sono quelli, il font e' quell'altro, sto costruendo la pagina contatti, l'ultima volta avevamo deciso di usare quel layout...

Con il sistema, lanci `python memoria.py start`. Lo script ti risponde:

```
FILE NUOVI: contatti.html (3.2 KB) FILE MODIFICATI: style.css (4.1 KB → 5.3 KB) Resto:
28 file invariati.
```

Trascini `CONTEXT.md`. Scrivi `START`. Claude risponde:

> Allineato. Hai aggiunto contatti.html e modificato style.css. Cosa facciamo?

Sa gia' tutto. Hai scritto cinque parole.

Hai bisogno di una mano?

Se qualcosa non gira o vuoi un parere sull'adattamento al tuo progetto specifico, scrivimi. Mi interessa raccogliere i punti di attrito reali per migliorare la guida.

Vuoi capire meglio la filosofia dietro al sistema?

Leggi l'articolo completo: **Context Engineering: come ho costruito un sistema di memoria per i progetti AI**. Dentro trovi il ragionamento, l'ispirazione di Karpathy e i risultati misurati.

Scritto da **Nicola Silvestre** — nicolasilvestre.it — Aprile 2026