CLI, headless e automazione ◆

Claude come strumento scriptabile: comandi e flag di lancio, modalità non interattiva (-p), output strutturato, cron, CI/CD e routine cloud.

Aggiornato: luglio 2026 · Fonte: documentazione ufficiale Anthropic (code.claude.com/docs) · Vol. 4 della serie bignami Claude Code.

1. Comandi CLI

quello che digiti prima di entrare in sessione

Comando	Cosa fa
`claude` / `claude "query"`	Sessione interattiva, con prompt iniziale opzionale.
`claude -p "query"`	**Non interattivo**: esegue e esce. La base di ogni scripting (vedi cap. 3).
`claude -c` / `claude -r "id" "query"`	Continua l'ultima conversazione nella directory / riprende per ID o nome.
`claude agents`	Apre l'**agent view**: dispatch e monitoraggio delle sessioni in background. `--json` per lo scripting, `--cwd <path>` per filtrare per progetto.
`claude attach / logs / stop / respawn / rm <id>`	Gestione sessioni background dalla shell: attacca, stampa output recente, ferma, riavvia con conversazione intatta, rimuove dalla lista.
`claude mcp add|list|get|remove`	Configura i server MCP; `claude mcp login <nome>` esegue l'OAuth senza aprire il pannello (con `--no-browser` per SSH).
`claude plugin install ...`	Gestione plugin da CLI (es. `claude plugin install code-review@claude-plugins-official`).
`claude setup-token`	Genera un **token OAuth a lunga vita per CI e script** (richiede abbonamento). Lo stampa senza salvarlo.
`claude project purge [path]`	Cancella lo stato locale di un progetto: transcript, log di debug, history. `--dry-run` per l'anteprima.
`claude update` / `claude install stable`	Aggiorna / installa una versione specifica del binario nativo.
`claude auth login|logout|status`	Autenticazione; `status` esce con 0 se loggato, 1 se no (comodo negli script).

2. I flag che contano

claude --help non li elenca tutti

Flag	Effetto
`--model / --permission-mode / --effort`	Modello, modalità permessi e livello di ragionamento della sessione.
`--allowedTools "..."`	Tool pre-approvati senza prompt, con la sintassi delle regole: `"Bash(git diff *)"` `"Read"`. Per *limitare* i tool disponibili c'è invece `--tools`.
`--add-dir <path>...`	Directory di lavoro aggiuntive (accesso ai file; la config .claude/ non viene scoperta lì, con l'eccezione delle skill).
`--agents '<json>'`	Subagenti definiti al volo in JSON, solo per la sessione: perfetti per script e test.
`--append-system-prompt "..."`	Accoda testo al system prompt di default (anche da file con `--append-system-prompt-file`); `--system-prompt` lo sostituisce del tutto.
`--bare`	**Modalità minimale**: salta hook, skill, plugin, MCP, auto memory e CLAUDE.md. Avvio più rapido e, soprattutto, **riproducibile**: stesso risultato su ogni macchina. Raccomandata per script e CI (diventerà il default di -p).
`--bg / --background`	Avvia come **agente in background** e ritorna subito, stampando ID e comandi di gestione.
`--output-format text|json|stream-json`	Formato dell'output in -p (vedi sotto).
`--settings / --mcp-config / --plugin-dir`	Iniettano configurazione esplicita (file o JSON): il modo per dare a un run --bare esattamente ciò che serve.
`--dangerously-skip-permissions`	Nessun prompt di permesso. Solo in ambienti isolati (container/VM); la variante `--allow-...` aggiunge la modalità al ciclo Shift+Tab senza partirci dentro.

3. Headless: claude -p

Claude come comando unix

`-p` legge anche lo **stdin** (fino a 10MB) e scrive su stdout: si incastra nelle pipe come qualsiasi altro strumento. Le skill invocabili funzionano anche qui: includi `/nome-skill` nella stringa del prompt e viene espansa prima dell'esecuzione.

```
# Pipe: spiega un errore di build
cat build-error.txt | claude -p \
  'spiega concisamente la causa' > out.txt

# Linter custom in package.json
"lint:claude": "git diff main | claude -p
  \"sei un typo linter. per ogni typo:
  file:riga e problema. nient'altro.\""

# Commit automatico con permessi mirati
claude -p "crea un commit appropriato
  dalle modifiche staged" \
  --allowedTools "Bash(git diff *),
  Bash(git status *),Bash(git commit *)"

# CI riproducibile: bare mode
claude --bare -p "riassumi questo file" \
  --allowedTools "Read"
```

```
# Output JSON con metadati e costi
claude -p "riassumi il progetto" \
  --output-format json | jq -r '.result'
# il payload include total_cost_usd

# Output conforme a uno schema
claude -p "estrai i nomi delle funzioni
  da auth.py" --output-format json \
  --json-schema '{"type":"object",
   "properties":{"functions":{"type":
   "array","items":{"type":"string"}}},
   "required":["functions"]}' \
  | jq '.structured_output'

# Streaming token per token
claude -p "spiega la ricorsione" \
  --output-format stream-json --verbose \
  --include-partial-messages
```

Il pattern giusto per gli script: --bare + flag espliciti

Senza --bare, `claude -p` carica lo stesso contesto di una sessione interattiva: un hook nel ~/.claude di un collega o un MCP nel .mcp.json del progetto girerebbero anche in CI. Con --bare parti pulito (Bash + lettura/scrittura file) e aggiungi solo l'esplicito: `--append-system-prompt(-file)` per le istruzioni, `--settings`, `--mcp-config`, `--agents`, `--plugin-dir`. L'autenticazione in bare mode arriva da `ANTHROPIC_API_KEY` (niente keychain); in CI con abbonamento usa il token di `claude setup-token`. Per baseline di permessi: `--permission-mode dontAsk` nega tutto ciò che non è pre-approvato (CI blindata), `acceptEdits` lascia scrivere file senza prompt.

4. Automazione ricorrente

cron, routine e CI

Cron locale (stile sysadmin)

```
# crontab: report giornaliero alle 7
0 7 * * * cd /srv/repo && claude --bare -p \
 "controlla i log in /var/log/app di ieri,
  riassumi errori e anomalie in
  /srv/reports/\$(date +\%F).md" \
 --allowedTools "Read,Write,Bash(grep *)" \
 --output-format json >> /var/log/claude-cron.log
```

Dentro una sessione aperta, l'equivalente è `/loop [intervallo] [prompt]` (es. */loop 5m controlla se il deploy è finito*).

Routine cloud: /schedule

Le **routine** girano su infrastruttura gestita Anthropic: a orario, via API o in reazione a eventi GitHub — senza tenere acceso nulla di tuo. Si creano conversazionalmente con `/schedule` (alias /routines); serve GitHub collegato via `/web-setup`.

CI/CD

- **GitHub Actions**: `/install-github-app` configura app, workflow e secrets; da lì Claude risponde a @claude nelle PR/issue e può fare review automatiche. Esiste l'equivalente **GitLab CI/CD**.
- **/autofix-pr**: sessione cloud che sorveglia la PR del branch e pusha fix quando la CI fallisce o arrivano commenti.
- In pipeline generiche: `claude --bare -p` + `--output-format json`; con stream-json l'evento `system/init` elenca i plugin caricati (utile per far fallire la CI se un plugin non si carica) e `system/api_retry` segnala i retry.

Dettagli che salvano ore

- Con -p, i processi Bash in background vengono terminati ~5s dopo il risultato finale; i subagenti in background invece vengono attesi (cap 10 minuti di default).
- Preferisci il pipe del diff al permesso Bash: `git diff | claude -p ...` non richiede alcun allowedTools.
- `Bash(git diff *)`: lo spazio prima di * è significativo — senza, matcha anche `git diff-index`.
- Per l'SDK completo (Python/TypeScript, structured outputs, approvazioni programmatiche): pacchetto Claude Agent SDK, stessa macchina di Claude Code come libreria.

Fonti: code.claude.com/docs — cli-reference, headless, github-actions, routines · claude --help non elenca tutti i flag: la reference online sì.