CLI, headless and automation ◆

Claude as a scriptable tool: commands and launch flags, non-interactive mode (-p), structured output, cron, CI/CD and cloud routines.

Updated: July 2026 · Source: official Anthropic documentation (code.claude.com/docs) · Vol. 4 of the Claude Code cheatsheet series.

1. CLI commands

what you type before entering a session

Command	What it does
`claude` / `claude "query"`	Interactive session, with an optional initial prompt.
`claude -p "query"`	**Non-interactive**: runs and exits. The foundation of all scripting (see ch. 3).
`claude -c` / `claude -r "id" "query"`	Continues the latest conversation in the directory / resumes by ID or name.
`claude agents`	Opens the **agent view**: dispatch and monitoring of background sessions. `--json` for scripting, `--cwd <path>` to filter by project.
`claude attach / logs / stop / respawn / rm <id>`	Manage background sessions from the shell: attach, print recent output, stop, restart with the conversation intact, remove from the list.
`claude mcp add|list|get|remove`	Configure MCP servers; `claude mcp login <name>` runs the OAuth flow without opening the panel (with `--no-browser` for SSH).
`claude plugin install ...`	Plugin management from the CLI (e.g. `claude plugin install code-review@claude-plugins-official`).
`claude setup-token`	Generates a **long-lived OAuth token for CI and scripts** (requires a subscription). Prints it without saving it.
`claude project purge [path]`	Deletes a project's local state: transcripts, debug logs, history. `--dry-run` for a preview.
`claude update` / `claude install stable`	Updates / installs a specific version of the native binary.
`claude auth login|logout|status`	Authentication; `status` exits with 0 if logged in, 1 if not (handy in scripts).

2. The flags that matter

claude --help doesn't list them all

Flag	Effect
`--model / --permission-mode / --effort`	Model, permission mode and reasoning effort for the session.
`--allowedTools "..."`	Tools pre-approved without prompts, using rule syntax: `"Bash(git diff *)" "Read"`. To *restrict* the available tools, use `--tools` instead.
`--add-dir <path>...`	Additional working directories (file access; the .claude/ config there is not discovered, with the exception of skills).
`--agents '<json>'`	Subagents defined on the fly in JSON, for the session only: perfect for scripts and tests.
`--append-system-prompt "..."`	Appends text to the default system prompt (also from a file with `--append-system-prompt-file`); `--system-prompt` replaces it entirely.
`--bare`	**Minimal mode**: skips hooks, skills, plugins, MCP, auto memory and CLAUDE.md. Faster startup and, above all, **reproducible**: same result on every machine. Recommended for scripts and CI (will become the default for -p).
`--bg / --background`	Launches as a **background agent** and returns immediately, printing the ID and management commands.
`--output-format text|json|stream-json`	Output format in -p (see below).
`--settings / --mcp-config / --plugin-dir`	Inject explicit configuration (file or JSON): the way to give a --bare run exactly what it needs.
`--dangerously-skip-permissions`	

No permission prompts. Only in isolated environments (container/VM); the `--allow-...` variant adds the mode to the Shift+Tab cycle without starting in it.

3. Headless: claude -p

Claude as a unix command

`-p` also reads **stdin** (up to 10MB) and writes to stdout: it slots into pipes like any other tool. Invocable skills work here too: include `/skill-name` in the prompt string and it gets expanded before execution.

```
# Pipe: explain a build error
cat build-error.txt | claude -p \
  'concisely explain the cause' > out.txt

# Custom linter in package.json
"lint:claude": "git diff main | claude -p
  \"you are a typo linter. for each typo:
  file:line and issue. nothing else.\""

# Automatic commit with targeted permissions
claude -p "create an appropriate commit
  from the staged changes" \
  --allowedTools "Bash(git diff *),
  Bash(git status *),Bash(git commit *)"

# Reproducible CI: bare mode
claude --bare -p "summarize this file" \
  --allowedTools "Read"
```

```
# JSON output with metadata and costs
claude -p "summarize the project" \
  --output-format json | jq -r '.result'
# the payload includes total_cost_usd

# Output conforming to a schema
claude -p "extract the function names
  from auth.py" --output-format json \
  --json-schema '{"type":"object",
   "properties":{"functions":{"type":
   "array","items":{"type":"string"}}},
   "required":["functions"]}' \
  | jq '.structured_output'

# Token-by-token streaming
claude -p "explain recursion" \
  --output-format stream-json --verbose \
  --include-partial-messages
```

The right pattern for scripts: --bare + explicit flags

Without --bare, `claude -p` loads the same context as an interactive session: a hook in a colleague's ~/.claude or an MCP in the project's .mcp.json would also run in CI. With --bare you start clean (Bash + file read/write) and add only what's explicit: `--append-system-prompt(-file)` for instructions, `--settings`, `--mcp-config`, `--agents`, `--plugin-dir`. Authentication in bare mode comes from `ANTHROPIC_API_KEY` (no keychain); in CI with a subscription, use the token from `claude setup-token`. For permission baselines: `--permission-mode dontAsk` denies everything not pre-approved (locked-down CI), `acceptEdits` lets it write files without prompts.

4. Recurring automation

cron, routines and CI

Local cron (sysadmin style)

```
# crontab: daily report at 7am
0 7 * * * cd /srv/repo && claude --bare -p \
 "check yesterday's logs in /var/log/app,
  summarize errors and anomalies in
  /srv/reports/\$(date +\%F).md" \
 --allowedTools "Read,Write,Bash(grep *)" \
 --output-format json >> /var/log/claude-cron.log
```

Inside an open session, the equivalent is `/loop [interval] [prompt]` (e.g. */loop 5m check whether the deploy has finished*).

Cloud routines: /schedule

Routines run on Anthropic-managed infrastructure: on a schedule, via API or in reaction to GitHub events — without keeping anything of yours running. Create them conversationally with `/schedule` (alias /routines); requires GitHub connected via `/web-setup`.

CI/CD

- **GitHub Actions**: `/install-github-app` sets up the app, workflow and secrets; from there Claude responds to @claude in PRs/issues and can run automatic reviews. A **GitLab CI/CD** equivalent exists.
- **/autofix-pr**: a cloud session that watches the branch's PR and pushes fixes when CI fails or comments come in.
- In generic pipelines: `claude --bare -p` + `--output-format json`; with stream-json the `system/init` event lists the loaded plugins (useful for failing CI if a plugin doesn't load) and `system/api_retry` reports retries.

Details that save hours

- With -p, background Bash processes are killed ~5s after the final result; background subagents are awaited instead (10-minute cap by default).
- Prefer piping the diff over granting Bash permission: `git diff | claude -p ...` requires no allowedTools at all.
- `Bash(git diff *)`: the space before * is significant — without it, it also matches `git diff-index`.
- For the full SDK (Python/TypeScript, structured outputs, programmatic approvals): the Claude Agent SDK package, the same engine as Claude Code as a library.

Sources: code.claude.com/docs — cli-reference, headless, github-actions, routines · claude --help doesn't list all the flags: the online reference does.