Skills

Slash Commands

Commands are handled by the Gateway. Most commands must be sent as a **standalone** message that starts with `/`. The host-only bash chat command uses `! <cmd>` (with `/bash <cmd>` as an alias).

There are two related systems:

- **Commands**: standalone `/...` messages.
- **Directives**: `/think`, `/verbose`, `/reasoning`, `/elevated`, `/exec`, `/model`, `/queue`.
  - Directives are stripped from the message before the model sees it.
  - In normal chat messages (not directive-only), they are treated as “inline hints” and do **not** persist session settings.
  - In directive-only messages (the message contains only directives), they persist to the session and reply with an acknowledgement.
  - Directives are only applied for **authorized senders**. If `commands.allowFrom` is set, it is the only allowlist used; otherwise authorization comes from channel allowlists/pairing plus `commands.useAccessGroups`. Unauthorized senders see directives treated as plain text.

There are also a few **inline shortcuts** (allowlisted/authorized senders only): `/help`, `/commands`, `/status`, `/whoami` (`/id`). They run

immediately, are stripped before the model sees the message, and the remaining text continues through the normal flow.

Config

```
{
  commands: {
    native: "auto",
    nativeSkills: "auto",
    text: true,
    bash: false,
    bashForegroundMs: 2000,
    config: false,
    debug: false,
    restart: false,
    allowFrom: {
      "*": ["user1"],
      discord: ["user:123"],
    },
    useAccessGroups: true,
  },
}
```

`commands.text` (default `true`) enables parsing `/...` in chat messages.

On surfaces without native commands (WhatsApp/WebChat/Signal/iMessage/Google Chat/MS Teams), text commands still work even if you set this to `false`.

`commands.native` (default `"auto"`) registers native commands.

Auto: on for Discord/Telegram; off for Slack (until you add slash commands); ignored for providers without native support.

Set `channels.discord.commands.native`, `channels.telegram.commands.native`, or `channels.slack.commands.native` to override per provider (bool or `"auto"`).

`false` clears previously registered commands on Discord/Telegram at startup. Slack commands are managed in the Slack app and are not removed automatically.

`commands.nativeSkills` (default `"auto"`) registers **skill** commands natively when supported.

Auto: on for Discord/Telegram; off for Slack (Slack requires creating a slash command per skill).

Set `channels.discord.commands.nativeSkills`, `channels.telegram.commands.nativeSkills`, or `channels.slack.commands.nativeSkills` to override per provider (bool or `"auto"`).

`commands.bash` (default `false`) enables `! <cmd>` to run host shell commands (`/bash <cmd>` is an alias; requires `tools.elevated` allowlists).

`commands.bashForegroundMs` (default `2000`) controls how long bash waits before switching to background mode (`0` backgrounds immediately).

`commands.config` (default `false`) enables `/config` (reads/writes `openclaw.json`).

`commands.debug` (default `false`) enables `/debug` (runtime-only overrides).

`commands.allowFrom` (optional) sets a per-provider allowlist for command authorization. When configured, it is the only authorization source for commands and directives (channel allowlists/pairing and `commands.useAccessGroups` are ignored). Use `"*"` for a global default; provider-specific keys override it.

`commands.useAccessGroups` (default `true`) enforces allowlists/policies for commands when `commands.allowFrom` is not set.

Command list

Text + native (when enabled):

`/help`

`/commands`

`/skill <name> [input]` (run a skill by name)

`/status` (show current status; includes provider usage/quota for the current model provider when available)

`/allowlist` (list/add/remove allowlist entries)

`/approve <id> allow-once|allow-always|deny` (resolve exec approval prompts)

`/context [list|detail|json]` (explain “context”; `detail` shows per-file + per-tool + per-skill + system prompt size)

`/export-session [path]` (alias: `/export`) (export current session to HTML with full system prompt)

`/whoami` (show your sender id; alias: `/id`)

`/subagents list|kill|log|info|send|steer|spawn` (inspect, control, or spawn sub-agent runs for the current session)

`/kill <id|#|all>` (immediately abort one or all running sub-agents for this session; no confirmation message)

`/steer <id|#> <message>` (steer a running sub-agent immediately: in-run when possible, otherwise abort current work and restart on the steer message)

`/tell <id|#> <message>` (alias for `/steer`)

`/config show|get|set|unset` (persist config to disk, owner-only; requires `commands.config: true`)

`/debug show|set|unset|reset` (runtime overrides, owner-only; requires `commands.debug: true`)

`/usage off|tokens|full|cost` (per-response usage footer or local cost summary)

- `/tts off|always|inbound|tagged|status|provider|limit|summary|audio` (control TTS; see **/tts**)

  > Discord: native command is `/voice` (Discord reserves `/tts`); text `/tts` still works.

- `/stop`
- `/restart`
- `/dock-telegram` (alias: `/dock_telegram`) (switch replies to Telegram)
- `/dock-discord` (alias: `/dock_discord`) (switch replies to Discord)
- `/dock-slack` (alias: `/dock_slack`) (switch replies to Slack)
- `/activation mention|always` (groups only)
- `/send on|off|inherit` (owner-only)
- `/reset` or `/new [model]` (optional model hint; remainder is passed through)
- `/think <off|minimal|low|medium|high|xhigh>` (dynamic choices by model/provider; aliases: `/thinking`, `/t`)
- `/verbose on|full|off` (alias: `/v`)
- `/reasoning on|off|stream` (alias: `/reason`; when on, sends a separate message prefixed `Reasoning:`; `stream` = Telegram draft only)
- `/elevated on|off|ask|full` (alias: `/elev`; `full` skips exec approvals)
- `/exec host=<sandbox|gateway|node> security=<deny|allowlist|full> ask=<off|on-miss|always> node=<id>` (send `/exec` to show current)
- `/model <name>` (alias: `/models`; or `/<alias>` from `agents.defaults.models.*.alias`)
- `/queue <mode>` (plus options like `debounce:2s cap:25 drop:summarize`; send `/queue` to see current settings)
- `/bash <command>` (host-only; alias for `! <command>`; requires `commands.bash: true` + `tools.elevated` allowlists)

Text-only:

`/compact [instructions]` (see **/concepts/compaction**)

`! <command>` (host-only; one at a time; use `!poll` + `!stop` for long-running jobs)

`!poll` (check output / status; accepts optional `sessionId`; `/bash poll` also works)

`!stop` (stop the running bash job; accepts optional `sessionId`; `/bash stop` also works)

Notes:

Commands accept an optional `:` between the command and args (e.g. `/think: high`, `/send: on`, `/help:`).

`/new <model>` accepts a model alias, `provider/model`, or a provider name (fuzzy match); if no match, the text is treated as the message body.

For full provider usage breakdown, use `openclaw status --usage`.

`/allowlist add|remove` requires `commands.config=true` and honors channel `configWrites`.

`/usage` controls the per-response usage footer; `/usage cost` prints a local cost summary from OpenClaw session logs.

`/restart` is disabled by default; set `commands.restart: true` to enable it.

`/verbose` is meant for debugging and extra visibility; keep it **off** in normal use.

`/reasoning` (and `/verbose`) are risky in group settings: they may reveal internal reasoning or tool output you did not intend to expose. Prefer leaving them off, especially in group chats.

Fast path: command-only messages from allowlisted senders are handled immediately (bypass queue + model).

Group mention gating: command-only messages from allowlisted senders bypass mention requirements.

Inline shortcuts (allowlisted senders only): certain commands also work when embedded in a normal message and are stripped before the model sees the remaining text.

Example: `hey /status` triggers a status reply, and the remaining text continues through the normal flow.

Currently: `/help`, `/commands`, `/status`, `/whoami` (`/id`).

Unauthorized command-only messages are silently ignored, and inline `/...` tokens are treated as plain text.

Skill commands: `user-invocable` skills are exposed as slash commands. Names are sanitized to `a-z0-9_` (max 32 chars); collisions get numeric suffixes (e.g. `_2`).

`/skill <name> [input]` runs a skill by name (useful when native command limits prevent per-skill commands).

By default, skill commands are forwarded to the model as a normal request.

Skills may optionally declare `command-dispatch: tool` to route the command directly to a tool (deterministic, no model).

Example: `/prose` (OpenProse plugin) — see **OpenProse**.

Native command arguments: Discord uses autocomplete for dynamic options (and button menus when you omit required args). Telegram and Slack show a button menu when a command supports choices and you omit the arg.

Usage surfaces (what shows where)

Provider usage/quota (example: “Claude 80% left”) shows up in `/status` for the current model provider when usage tracking is enabled.

Per-response tokens/cost is controlled by `/usage off|tokens|full` (appended to normal replies).

`/model status` is about **models/auth/endpoints**, not usage.

Model selection (`/model`)

`/model` is implemented as a directive.

Examples:

```
/model
/model list
/model 3
/model openai/gpt-5.2
/model opus@anthropic:default
/model status
```

Notes:

- `/model` and `/model list` show a compact, numbered picker (model family + available providers).
- `/model <#>` selects from that picker (and prefers the current provider when possible).
- `/model status` shows the detailed view, including configured provider endpoint (`baseUrl`) and API mode (`api`) when available.

Debug overrides

`/debug` lets you set **runtime-only** config overrides (memory, not disk). Owner-only. Disabled by default; enable with `commands.debug: true` .

Examples:

```
/debug show
/debug set messages.responsePrefix="[openclaw]"
/debug set channels.whatsapp.allowFrom=["+1555","+4477"]
/debug unset messages.responsePrefix
/debug reset
```

Notes:

- Overrides apply immediately to new config reads, but do **not** write to `openclaw.json`.
- Use `/debug reset` to clear all overrides and return to the on-disk config.

Config updates

`/config` writes to your on-disk config (`openclaw.json`). Owner-only. Disabled by default; enable with `commands.config: true`.

Examples:

```
/config show
/config show messages.responsePrefix
/config get messages.responsePrefix
/config set messages.responsePrefix="[openclaw]"
/config unset messages.responsePrefix
```

Notes:

- Config is validated before write; invalid changes are rejected.
- `/config` updates persist across restarts.

Surface notes

Text commands run in the normal chat session (DMs share `main`, groups have their own session).

Native commands use isolated sessions:

- Discord: `agent:<agentId>:discord:slash:<userId>`
- Slack: `agent:<agentId>:slack:slash:<userId>` (prefix configurable via `channels.slack.slashCommand.sessionPrefix`)
- Telegram: `telegram:slash:<userId>` (targets the chat session via `CommandTargetSessionKey`)

`/stop` targets the active chat session so it can abort the current run.

Slack: `channels.slack.slashCommand` is still supported for a single `/openclaw`-style command. If you enable `commands.native`, you must create one Slack slash command per built-in command (same names as `/help`). Command argument menus for Slack are delivered as ephemeral Block Kit buttons.

‹ Multi-Agent Sandbox & Tools Skills ›