Messaging platforms

Discord

Status: ready for DMs and guild channels via the official Discord gateway.

Pairing
Discord DMs default to pairing mode.

Slash commands
Native command behavior and command catalog.

Channel troubleshooting
Cross-channel diagnostics and repair flow.

Quick setup

You will need to create a new application with a bot, add the bot to your server, and pair it to OpenClaw. We recommend adding your bot to your own private server. If you don't have one yet, **create one first** (choose **Create My Own > For me and my friends**).

1 **Create a Discord application and bot**

Go to the **Discord Developer Portal** and click **New Application**. Name it something like "OpenClaw".

Click **Bot** on the sidebar. Set the **Username** to whatever you call your OpenClaw agent.

2 Enable privileged intents

Still on the **Bot** page, scroll down to **Privileged Gateway Intents** and enable:

Message Content Intent (required)

Server Members Intent (recommended; required for role allowlists and name-to-ID matching)

Presence Intent (optional; only needed for presence updates)

3 Copy your bot token

Scroll back up on the **Bot** page and click **Reset Token**.

> Despite the name, this generates your first token — nothing is being “reset.”

Copy the token and save it somewhere. This is your **Bot Token** and you will need it shortly.

4 Generate an invite URL and add the bot to your server

Click **OAuth2** on the sidebar. You’ll generate an invite URL with the right permissions to add the bot to your server.

Scroll down to **OAuth2 URL Generator** and enable:

`bot`

`applications.commands`

A **Bot Permissions** section will appear below. Enable:

View Channels

Send Messages

Read Message History

Embed Links

Attach Files

Add Reactions (optional)

Copy the generated URL at the bottom, paste it into your browser, select your server, and click **Continue** to connect. You should now see your bot in the Discord server.

5 **Enable Developer Mode and collect your IDs**

Back in the Discord app, you need to enable Developer Mode so you can copy internal IDs.

1. Click **User Settings** (gear icon next to your avatar) → **Advanced** → toggle on **Developer Mode**
2. Right-click your **server icon** in the sidebar → **Copy Server ID**
3. Right-click your **own avatar** → **Copy User ID**

Save your **Server ID** and **User ID** alongside your Bot Token — you'll send all three to OpenClaw in the next step.

6 **Allow DMs from server members**

For pairing to work, Discord needs to allow your bot to DM you. Right-click your **server icon** → **Privacy Settings** → toggle on **Direct Messages**.

This lets server members (including bots) send you DMs. Keep this enabled if you want to use Discord DMs with OpenClaw. If you only plan to use guild channels, you can disable DMs after pairing.

7 **Step 0: Set your bot token securely (do not send it in chat)**

Your Discord bot token is a secret (like a password). Set it on the machine running OpenClaw before messaging your agent.

```
openclaw config set channels.discord.token '"YOUR_BOT_TOKEN"' --
openclaw config set channels.discord.enabled true --json
openclaw gateway
```

If OpenClaw is already running as a background service, use `openclaw gateway restart` instead.

8 Configure OpenClaw and pair

Ask your agent | CLI / config

Chat with your OpenClaw agent on any existing channel (e.g. Telegram) and tell it. If Discord is your first channel, use the CLI / config tab instead.

> "I already set my Discord bot token in config. Please finish Discord setup with User ID `<user_id>` and Server ID `<server_id>`."

9 Approve first DM pairing

Wait until the gateway is running, then DM your bot in Discord. It will respond with a pairing code.

Ask your agent | CLI

Send the pairing code to your agent on your existing channel:

> "Approve this Discord pairing code: `<CODE>`"

Pairing codes expire after 1 hour.

You should now be able to chat with your agent in Discord via DM.

Token resolution is account-aware. Config token values win over env fallback. `DISCORD_BOT_TOKEN` is only used for the default account.

Recommended: Set up a guild workspace

Once DMs are working, you can set up your Discord server as a full workspace where each channel gets its own agent session with its own context. This is recommended for private servers where it's just you and your bot.

1 Add your server to the guild allowlist

This enables your agent to respond in any channel on your server, not just DMs.

Ask your agent | Config

> "Add my Discord Server ID `<server_id>` to the guild allowlist"

2 Allow responses without @mention

By default, your agent only responds in guild channels when @mentioned. For a private server, you probably want it to respond to every message.

Ask your agent | Config

> "Allow my agent to respond on this server without having to be @mentioned"

Plan for memory in guild channels

By default, long-term memory (MEMORY.md) only loads in DM sessions. Guild channels do not auto-load MEMORY.md.

Ask your agent | Manual

> "When I ask questions in Discord channels, use memory_search or memory_get if you need long-term context from MEMORY.md."

Now create some channels on your Discord server and start chatting. Your agent can see the channel name, and each channel gets its own isolated session — so you can set up `#coding`, `#home`, `#research`, or whatever fits your workflow.

Runtime model

- Gateway owns the Discord connection.
- Reply routing is deterministic: Discord inbound replies back to Discord.
- By default (`session.dmScope=main`), direct chats share the agent main session (`agent:main:main`).
- Guild channels are isolated session keys (`agent:<agentId>:discord:channel:<channelId>`).
- Group DMs are ignored by default (`channels.discord.dm.groupEnabled=false`).
- Native slash commands run in isolated command sessions (`agent:<agentId>:discord:slash:<userId>`), while still carrying `CommandTargetSessionKey` to the routed conversation session.

Interactive components

OpenClaw supports Discord components v2 containers for agent messages. Use the message tool with a `components` payload. Interaction results are routed back to the agent as normal inbound messages and follow the existing Discord `replyToMode` settings.

Supported blocks:

- `text`, `section`, `separator`, `actions`, `media-gallery`, `file`
  - Action rows allow up to 5 buttons or a single select menu
  - Select types: `string`, `user`, `role`, `mentionable`, `channel`

By default, components are single use. Set `components.reusable=true` to allow buttons, selects, and forms to be used multiple times until they expire.

To restrict who can click a button, set `allowedUsers` on that button (Discord user IDs, tags, or `*`). When configured, unmatched users receive an ephemeral denial.

File attachments:

- `file` blocks must point to an attachment reference (`attachment://<filename>`)
  - Provide the attachment via `media` / `path` / `filePath` (single file); use `media-gallery` for multiple files
  - Use `filename` to override the upload name when it should match the attachment reference

Modal forms:

- Add `components.modal` with up to 5 fields
  - Field types: `text`, `checkbox`, `radio`, `select`, `role-select`, `user-select`
  - OpenClaw adds a trigger button automatically

Example:

```
channel: "discord",
action: "send",
to: "channel:123456789012345678",
message: "Optional fallback text",
components: {
  reusable: true,
  text: "Choose a path",
  blocks: [
    {
      type: "actions",
      buttons: [
        {
          label: "Approve",
          style: "success",
          allowedUsers: ["123456789012345678"],
        },
        { label: "Decline", style: "danger" },
      ],
    },
    {
      type: "actions",
      select: {
        type: "string",
        placeholder: "Pick an option",
        options: [
          { label: "Option A", value: "a" },
          { label: "Option B", value: "b" },
        ],
      },
    },
  ],
  modal: {
    title: "Details",
    triggerLabel: "Open form",
    fields: [
      { type: "text", label: "Requester" },
      {
        type: "select",
        label: "Priority",
```

```
          options: [
            { label: "Low", value: "low" },
            { label: "High", value: "high" },
          ],
        },
      ],
    },
  },
}
```

Access control and routing

DM policy | Guild policy | Mentions and group DMs

`channels.discord.dmPolicy` controls DM access (legacy: `channels.discord.dm.policy`):

- `pairing` (default)
- `allowlist`
- `open` (requires `channels.discord.allowFrom` to include `"*"`; legacy: `channels.discord.dm.allowFrom`)
- `disabled`

If DM policy is not open, unknown users are blocked (or prompted for pairing in `pairing` mode).

DM target format for delivery:

- `user:<id>`
- `<@id>` mention

Bare numeric IDs are ambiguous and rejected unless an explicit user/channel target kind is provided.

Role-based agent routing

Use `bindings[].match.roles` to route Discord guild members to different agents by role ID. Role-based bindings accept role IDs only and are evaluated after peer or parent-peer bindings and before guild-only bindings. If a binding also sets other match fields (for example `peer` + `guildId` + `roles`), all configured fields must match.

```
{
  bindings: [
    {
      agentId: "opus",
      match: {
        channel: "discord",
        guildId: "123456789012345678",
        roles: ["111111111111111111"],
      },
    },
    {
      agentId: "sonnet",
      match: {
        channel: "discord",
        guildId: "123456789012345678",
      },
    },
  ],
}
```

Developer Portal setup

Create app and bot
Privileged intents
OAuth scopes and baseline permissions

Copy IDs

›

Native commands and command auth

`commands.native` defaults to `"auto"` and is enabled for Discord.

Per-channel override: `channels.discord.commands.native`.

`commands.native=false` explicitly clears previously registered Discord native commands.

Native command auth uses the same Discord allowlists/policies as normal message handling.

Commands may still be visible in Discord UI for users who are not authorized; execution still enforces OpenClaw auth and returns “not authorized”.

See **Slash commands** for command catalog and behavior.

Feature details

Reply tags and native replies
History, context, and thread behavior
Reaction notifications
Ack reactions
Config writes
Gateway proxy
PluralKit support

Presence configuration

Exec approvals in Discord ›

Tools and action gates

Discord message actions include messaging, channel admin, moderation, presence, and metadata actions.

Core examples:

- messaging: `sendMessage`, `readMessages`, `editMessage`, `deleteMessage`, `threadReply`
- reactions: `react`, `reactions`, `emojiList`
- moderation: `timeout`, `kick`, `ban`
- presence: `setPresence`

Action gates live under `channels.discord.actions.*`.

Default gate behavior:

Action group	Default
reactions, messages, threads, pins, polls, search, memberInfo, roleInfo, channelInfo, channels, voiceStatus, events, stickers, emojiUploads, stickerUploads, permissions	enabled
roles	disabled
moderation	disabled
presence	disabled

Components v2 UI

OpenClaw uses Discord components v2 for exec approvals and cross-context markers. Discord message actions can also accept `components` for custom UI (advanced; requires Carbon component instances), while legacy `embeds` remain available but are not recommended.

- `channels.discord.ui.components.accentColor` sets the accent color used by Discord component containers (hex).
- Set per account with `channels.discord.accounts.<id>.ui.components.accentColor`.
- `embeds` are ignored when components v2 are present.

Example:

```
{
  channels: {
    discord: {
      ui: {
        components: {
          accentColor: "#5865F2",
        },
      },
    },
  },
}
```

Voice messages

Discord voice messages show a waveform preview and require OGG/Opus audio plus metadata. OpenClaw generates the waveform automatically, but it needs `ffmpeg` and `ffprobe` available on the gateway host to inspect and convert audio files.

Requirements and constraints:

- Provide a **local file path** (URLs are rejected).

Omit text content (Discord does not allow text + voice message in the same payload).

Any audio format is accepted; OpenClaw converts to OGG/Opus when needed.

Example:

```
message(action="send", channel="discord", target="channel:123", path='
```

Troubleshooting

Used disallowed intents or bot sees no guild messages
Guild messages blocked unexpectedly
Require mention false but still blocked
Permissions audit mismatches
DM and pairing issues
Bot to bot loops

Configuration reference pointers

Primary reference:

High-signal Discord fields:

- startup/auth: `enabled` , `token` , `accounts.*` , `allowBots`
- policy: `groupPolicy` , `dm.*` , `guilds.*` , `guilds.*.channels.*`

command: `commands.native` , `commands.useAccessGroups` , `configWrites`

reply/history: `replyToMode` , `historyLimit` , `dmHistoryLimit` , `dms.*.historyLimit`

delivery: `textChunkLimit` , `chunkMode` , `maxLinesPerMessage`

media/retry: `mediaMaxMb` , `retry`

actions: `actions.*`

presence: `activity` , `status` , `activityType` , `activityUrl`

UI: `ui.components.accentColor`

features: `pluralkit` , `execApprovals` , `intents` , `agentComponents` , `heartbeat` , `responsePrefix`

Safety and operations

Treat bot tokens as secrets (`DISCORD_BOT_TOKEN` preferred in supervised environments).

Grant least-privilege Discord permissions.

If command deploy/state is stale, restart gateway and re-check with `openclaw channels status --probe` .

Related

Pairing

Channel routing

Multi-agent routing

Troubleshooting

Slash commands

‹ Telegram IRC ›

Powered by mintlify

›