

Your jig in git

Push a jig to your own GitHub, GitLab or other git repository, install it on your other machines and your teammates', and keep every copy up to date — without writing a script or installing git.

development · 2026-11-08

Your jig in git

Git carries the files; Ozwala does the installing. Push a jig to a repository of your own, install it on your next machine or a teammate's, and every copy keeps up: its data is reachable over MCP and its skills are in each agent, with nothing to write by hand. Ozwala does the git work itself, so you don't need git installed — though a jig is an ordinary repository, and your own git commands work in it too.

Push a jig

The first push needs somewhere to go.

- **A new repository on GitHub or GitLab.** Say which service, give a token, and create it:

  ```
  ozwala jig set my-jig --provider github --token-stdin
  ozwala jig push my-jig --create
  ```

  The repository is private unless you add `--public`. `--owner team` creates it in an organisation or group. On your own GitLab, also give its API: `--provider gitlab --api https://gitlab.example.com/api/v4`.

- **A repository you already have**, on any git service, over HTTPS or SSH:

  ```
  ozwala jig set my-jig --url git@git.example.com:me/my-jig.git
  ozwala jig push my-jig
  ```

After that, `ozwala jig push my-jig -m "What changed"` commits everything in the jig and sends it. In the desktop app, the jig's **Repository** panel holds the same settings, and **Push...** does the rest, including **Create repository and push**.

What goes into the repository

Everything in the jig directory, its data included: `data/vectors.db` (the search index) and `data/archive.db` (imported tables). A jig doesn't keep the documents it was built from, so the data is what lets another machine search it. A new jig gets a `.gitignore` that leaves out only SQLite's scratch files (`*.db-wal`, `*.db-shm`, `*.db-journal`) and system litter.

Two checks happen before anything is sent:

- **A public repository asks first.** If GitHub or GitLab says the repository is public and the push carries data, Ozwala asks before sending it (`--yes` answers for a script).

- **No file over 100 MB.** GitHub refuses such files, and so does GitLab by default; Ozwala refuses them before sending anything rather than failing halfway.

The index records which embedding model built it. A machine with another embedding model is told to switch, instead of getting wrong answers.

Install it elsewhere

```
ozwala jig install https://github.com/me/my-jig.git
```

A private repository asks for a token (`--token-stdin` for scripts) and saves it for the host. An SSH address uses your SSH keys. The jig's skills are copied into every agent as soon as it is installed. In the app: **New jig → Install one that already exists**; for a private repository it asks for the token there. **Copy install command** on the jig's screen gives you the line to send a teammate.

Keep every copy up to date

An installed jig follows its repository's **releases**: the newest version tag. Before a repository's first release it follows the latest commit, and says so. `ozwala jig set my-jig --follow commits` follows every push instead.

- **Automatically**, by default: when the app starts, every six hours while it is open, and when an agent starts `ozwala mcp`. A copy with changes of its own, an open proposal, or a pin is never touched. Turn it off for one jig with `ozwala jig set my-jig --auto-update off`, or for the whole machine with `auto_update: false` at the top of `config.yml`.
- **When you ask**: `ozwala jig update my-jig`, or `--all`. Its skills are re-synced to every agent.
- **Pin** a version to stay on it — `ozwala jig pin my-jig v1.2.0` — and `ozwala jig unpin my-jig` to follow releases again. Pinning an earlier version and updating is how you go back to it.

An update never throws away work. If a copy has edits or commits that aren't on the repository, the update refuses and says to push them, or to discard them with `--discard`. If someone else pushed since your last update, your push is refused and tells you to update first.

Release a version

```
ozwala jig release my-jig --bump minor --notes "Answers cite their sources"
```

This writes the new version into `jig.yml`, commits and tags it (`v1.3.0`), and pushes both. On GitHub and GitLab it also makes a release page, with your notes or the commits since the last release. Copies that follow releases move to it on their next update. In the app: **Release…** shows the version each kind of release would make.

Work with a team

Everyone uses their own token or SSH key, and the repository's access rules are the team's. A teammate with write access can **propose** a change instead of pushing it:

```
ozwala jig push my-jig --request --title "Add a churn question"
```

The change goes to a branch named after them and the title (`sam/add-a-churn-question`), and a pull request (GitHub) or merge request (GitLab) is opened. Later pushes add to it. Once it is merged, the next update puts the copy back on the main branch; if it was closed instead, the update offers to drop it. A request that changes `data/` says so: databases can't be merged, so merging it replaces the data.

Tokens and SSH

Tokens are saved in `credentials.yml` beside `config.yml`, readable only by you, and never shown again.

- For one jig: `ozwala jig set my-jig --token-stdin`.
- For every jig on a host: `ozwala git-token set github.com`, or **Settings → Git** in the app. `ozwala git-token list` shows the hosts, never the tokens.
- From the environment: `OZWALA_GIT_TOKEN`, `GH_TOKEN` (GitHub) or `GITLAB_TOKEN` (GitLab).

A GitHub token needs the `repo` scope (classic) or Contents and Administration write (fine-grained) to create repositories; a GitLab token needs `api`.

SSH uses your SSH agent first, then `~/.ssh/id_ed25519`, `id_ecdsa` or `id_rsa` without a passphrase (a key with a passphrase goes in the agent).

- **Linux and macOS:** the agent in `SSH_AUTH_SOCK`; on macOS, add a key with `ssh-add --apple-use-keychain`.
- **Windows:** Pageant, or the Windows OpenSSH agent service with the key added by `ssh-add`. (Git for Windows' own ssh-agent is not supported.)

A server must be in `~/.ssh/known_hosts`. An unknown one is refused, with its key's fingerprint and the `ssh-keyscan` line that adds it once you have checked it.