Wrangler Cloudflare Workers CLI

START HERE

`npm create cloudflare@latest`
Scaffold a Worker

`wrangler login`
Sign in to Cloudflare

`wrangler dev`
Run locally

`wrangler deploy`
Publish to the edge

`wrangler tail`
Stream live logs

`wrangler secret put <KEY>`
Store a secret

`wrangler d1 migrations apply <db>`
Update the database

`wrangler rollback`
Undo a bad deploy

Code + config src/ and wrangler.jsonc → `wrangler dev` → **Local preview** localhost:8787 → `wrangler deploy` → **Cloudflare edge** live Worker → `wrangler tail` → **Live logs** debug and watch

CONCEPTS

Wrangler is the Workers CLI. Run it per project with npx wrangler (or pnpm/yarn) so the version matches the repo.

Bindings (KV, R2, D1) are declared in the config file and reach your code as env.<NAME>.

Most storage commands take --binding <name> or an ID, and --local or --remote to pick where they act.

ACCOUNT

`wrangler login`
Sign in through the browser

`wrangler whoami`
Show the signed-in account

`wrangler logout`
Remove stored credentials

PROJECT

`npm create cloudflare@latest`
Create a new Worker project

`wrangler init [<name>]`
Set up Wrangler in a folder

`wrangler types`
Generate types for your bindings
Writes worker-configuration.d.ts; rerun after config changes

DEVELOP

`wrangler dev`
Run the Worker locally

`wrangler dev --port <n>`
Use another local port

`wrangler dev --remote`
Run against remote resources

`wrangler dev --local`
Keep all bindings local

DEPLOY

`wrangler deploy`
Publish the Worker

`wrangler deploy --env <name>`
Deploy an environment

`wrangler deploy --dry-run --outdir <dir>`
Build without publishing

`wrangler deploy --keep-vars`
Keep dashboard-set variables

`wrangler delete [<name>]`
Remove a Worker

LOGS

`wrangler tail`
Stream live logs

`wrangler tail --format pretty`
Human-readable log lines

`wrangler tail --status error`
Only failed requests

`wrangler tail --search <text>`
Filter by log text

SECRETS

`wrangler secret put <KEY>`
Add or update a secret

`wrangler secret list`
List secret names

`wrangler secret delete <KEY>`
Remove a secret

`wrangler secret bulk <file>`
Set many secrets from JSON or .env

Local secrets go in .dev.vars (or .env); keep it out of git.

VERSIONS

`wrangler versions upload`
Upload without going live

`wrangler versions list`
List recent versions

`wrangler versions deploy <id>@<pct>`
Split traffic between versions

`wrangler deployments list`
Show deployment history

`wrangler rollback [<id>]`
Return to an earlier version
Without an id it goes back one version

KV

`wrangler kv namespace create <name>`
Create a namespace

`wrangler kv namespace list`
List namespaces

`wrangler kv key put <key> <value> --binding <b>`
Write a key

`wrangler kv key get <key> --binding <b>`
Read a key

`wrangler kv key list --binding <b>`
List keys

KV (cont.)

`wrangler kv key delete <key> --binding <b>`
Delete a key

`wrangler kv bulk put <file.json> --binding <b>`
Write many keys

R2

`wrangler r2 bucket create <name>`
Create a bucket

`wrangler r2 bucket list`
List buckets

`wrangler r2 object put <bucket>/<key> --file <path>`
Upload an object

`wrangler r2 object get <bucket>/<key> --file <path>`
Download an object

`wrangler r2 object delete <bucket>/<key>`
Delete an object

Object paths are <bucket>/<key>. Add --remote to act on the real bucket instead of local storage.

D1

`wrangler d1 create <name>`
Create a database

`wrangler d1 list`
List databases

`wrangler d1 execute <db> --command "<sql>"`
Run one SQL statement

`wrangler d1 execute <db> --file <file.sql>`
Run a SQL file

`wrangler d1 migrations create <db> <name>`
Add a migration file

`wrangler d1 migrations apply <db>`
Apply pending migrations

`wrangler d1 migrations list <db>`
Show unapplied migrations

`wrangler d1 export <db> --output <file.sql>`
Dump to a SQL file

`wrangler d1 time-travel restore <db> --timestamp <t>`
Restore to a past moment

D1 commands run locally unless you pass --remote; apply migrations locally first, then with --remote.

CONFIG

Config fields

Field	Purpose
`name`	Worker name
`main`	Entry file
`compatibility_date`	Runtime behaviour date
`compatibility_flags`	Opt-in runtime features
`vars`	Plain variables
`kv_namespaces`	KV bindings
`r2_buckets`	R2 bindings
`d1_databases`	D1 bindings
`routes`	Custom domains and routes
`assets`	Static asset directory
`observability`	Logs and traces
`env.<name>`	Per-environment overrides

Cloudflare recommends wrangler.jsonc for new projects; wrangler.toml still works.

vars and bindings are not inherited by [env.<name>]: repeat them in each environment.

Set compatibility_date to the day you start; bump it deliberately to adopt new runtime behaviour.

RECIPES

First deploy

```
npm create cloudflare@latest my-app
cd my-app
npx wrangler dev
npx wrangler deploy
```

Database from scratch

```
npx wrangler d1 create <db>
npx wrangler d1 migrations create <db> init
npx wrangler d1 migrations apply <db> --local
npx wrangler d1 migrations apply <db> --remote
```

Gradual rollout

```
npx wrangler versions upload
npx wrangler versions deploy <new>@10% <old>@90% -y
# watch errors, then shift traffic
npx wrangler versions deploy <new>@100% -y
```

Add a secret and watch it work

```
npx wrangler secret put API_TOKEN
npx wrangler tail --status error
```