

Rules for AI in your team: a template

What the agent must not do, what stops it and who owns the rules. Ready for your own repository in one afternoon.

For CTOs, tech leads and developers whose team already uses Claude Code, Codex, Cursor or Copilot. Everyone in their own way. This template turns that into one shared layer.

Today AI writes all of our code and a senior answers for the result. Since 2023 we have delivered 15+ production projects. This is the framework we use on them.

1. Why rules in text are not enough

You forbid the agent to open a file. It doesn't open it. It writes a script that prints it.

That isn't malice. The agent got a task and looks for a way. A sentence in the prompt is an obstacle, not a wall. So every rule below has a column most "AI policies" lack: what actually stops it.

Level	What it means	Strength
Just a sentence	A rule in CLAUDE.md or AGENTS.md. The agent follows it most of the time.	weakest
Setting or hook	The command is stopped before it runs.	medium
CI	The change can't reach the main branch until the gate is green.	strong
Lint or test	The build fails and the message says why. The agent reads it and fixes it itself.	strongest

A rule broken twice stops being just a sentence. Move it one level up.

2. Eight things the agent must not do

#	The agent must not	Why	What stops it
1	Connect to the production database	A bad query on production can't be undone. A read-only replica or a log export is enough for diagnostics.	setting (deny) + separate credentials
2	Read secrets (.env, keys, tokens)	Whatever the agent reads ends up in context, in logs and sometimes in a commit.	setting + secrets outside the repository
3	Use production cloud credentials	Same reason as the database, bigger blast radius.	setting (deny)
4	Run a deploy or a migration	What can't be undone goes through a human.	setting (ask) + environment approval in CI

#	The agent must not	Why	What stops it
5	Rewrite history or push to main	Force-push and <code>reset --hard</code> silently destroy someone else's work.	setting + hook + branch protection on the server
6	Write the test you grade it by	An agent that writes its own test also writes its own grade.	CI check + a rule for the reviewer
7	Review its own change	One model that writes and checks has the same blind spots twice.	required PR approval by a human
8	Put customer data into a prompt	Personal data doesn't belong in a tool you haven't approved for it.	just a sentence + team agreement

For number eight we honestly write "just a sentence". Nothing enforces it mechanically today. A rule that admits its weakness beats a rule that pretends to be strong.

How many of your rules are just a sentence? That's how many holes you have.

3. Where each rule belongs

The agent loads its rules file on every run. The more text, the worse it sticks to what matters. So, three levels:

1. **Repository root:** a short CLAUDE.md or AGENTS.md. What this repo is, how to run tests, what the agent must not do, what finished work looks like.
2. **Folder:** rules for one module in a file inside it. They load only when the agent works there.
3. **Lint or test:** the most important rules. Nobody has to read them; they stop a bad change on their own.

Using several tools? Keep the rules in AGENTS.md and point to it from CLAUDE.md with the line `@AGENTS.md`. Two copies drift apart before anyone notices.

File structure

```

CLAUDE.md or AGENTS.md    short: repo, commands, links
.claude/
  settings.json            the bans in machine form
  hooks/push-gate.sh       what settings can't express
  rules/
    00-never.md           8 bans + "what enforces it" column
    10-task.md            what a task needs before the agent starts
    20-gates.md           lint, typecheck, tests: blocking yes/no
    30-review.md          who reviews and what they get
    90-ownership.md       who owns the rules and how a new one is added

```

4. Patterns for your repository

The examples are for Claude Code. The principle (deny, hook, CI, lint) works for other tools too; only the syntax differs.

Bans in settings (items 1 to 5)

```
{
  "permissions": {
    "deny": [
      "Bash(psql:*)",
      "Bash(git push --force:*)",
      "Bash(git reset --hard:*)",
      "Read(.env.prod*)",
      "Read(**/*.pem)"
    ],
    "ask": [
      "Read(**/.env*)",
      "Bash(npx prisma migrate deploy:*)"
    ]
  }
}
```

Watch out: a rule matches the command text, not the intent. If the agent may run `cat`, then `cat .env` gets around the read ban. That's why secrets live outside the repository. Scripts load them and never print them.

Hook against pushing to main (item 5)

```
#!/bin/bash
# .claude/hooks/push-gate.sh (PreToolUse, Bash)
# exit 0 = allow, exit 2 = stop
CMD=$(jq -r '.tool_input.command // empty')
[[ "$CMD" != *"git push"* ]] && exit 0
case "$CMD" in
  *" main"*|" master"*)
    echo "Pushing to main is not allowed. Open a PR." >&2
    exit 2 ;;
esac
exit 0
```

Two mistakes silently switch a hook off. `|| true` after the hook turns exit 2 into 0. And `2>/dev/null` throws the message away, so the agent doesn't know why it failed. Test the hook once for real. Turn on branch protection on the server too, because a hook only runs where it's installed.

Tests and review (items 6 and 7)

- **A test must fail first.** On the original code. Otherwise it only proves the agent agrees with itself.
- **A human approves the tests from the task.** CI warns when a PR changes tests and code at once. The reviewer decides.
- **Whoever wrote it doesn't review it.** Run the review in a fresh context, ideally with another model. It gets the diff and your rules, not the conversation with the author.
- **CI must block.** A pipeline that only reports is decoration.

5. Where new rules come from

From review. A comment that comes up a second time isn't the developer's mistake. It's a hole in the rules.

How many times	What to do
First time	Fix it and move on
Second time	Write a rule for the agent
Third time	Write a lint or test that catches it on its own

One condition: review is written in the pull request, not only said on a call. Spoken review leaves nothing to pull a rule from.

6. Ownership and measurement

Rules without an owner go stale within months. The agent keeps obeying them even when they no longer apply. Copy this into [90-ownership.md](#) :

```
Owner: [name]
New rule: PR into .claude/rules/, approved by the owner.
Cleanup: first Monday of the month, 30 minutes.
Ladder: broken twice -> one level up
        sentence -> setting or hook -> CI -> lint
Monthly we record: how many rules are still just a sentence.
```

Three numbers worth tracking every month:

- **Review comments per PR.** If the rules work, they go down.
- **How often a gate stopped a change.** And how often the agent then fixed it itself.
- **How many rules are still just a sentence.** That's next month's to-do list.

Don't measure commits or lines of code. AI inflates those in a second.

7. Team worksheet (45 minutes)

Print it, put it on the table and fill it in together.

#	The agent never	What enforces it? (sentence / setting / CI / lint)	Owner
1	Production database, only a replica or export		
2	Secrets outside the agent's context		
3	Production cloud credentials		

#	The agent never	What enforces it? (sentence / setting / CI / lint)	Owner
4	Deploys and migrations only through a human		
5	Force-push and push to main		
6	The test it is graded by		
7	Reviewing its own change		
8	Customer data in a prompt		
9			
10			

Done when you can say, for every row, what enforces it and who owns it.

Want to set up the rules with someone who works with them every day?

In 15 minutes with Filip we go through your completed worksheet. We tell you which two rules we would move up first. No slides.

Book 15 minutes with Filip: digitaliko.com/contact

Digitaliko s.r.o. · www.digitaliko.com · Custom software in weeks. Yours from day one.