Skills, MCP, and Sub-agents

Teaching an agent your work — and pointing it at real data

SEAN DAVIS, MD, PHD · 2026-07-31

Where we left off

The first session argued that an agent is useful because it can **act** — read your files, run your tools, and check its own work.

This session is about the three things you add to it:

- **Skills** — your procedures, written down once
- **MCP** — connections to real data and real systems
- **Sub-agents** — parallel workers with their own context

…and then the part you cannot install: **how the two of you actually work.**

Part 1 — Skills

A skill is a folder

```
bioc-package-check/
├── SKILL.md        # required: frontmatter + instructions
├── scripts/        # optional: code the agent can run
├── references/     # optional: docs loaded on demand
└── assets/         # optional: templates, data
```

`SKILL.md` needs exactly two fields: **`name`** and **`description`**.

Everything else is optional.

Progressive disclosure — why skills are nearly free

Three stages, and only the first one is always paid for:

Stage	What loads	Cost
Discovery	`name` + `description`, for every installed skill	~100 tokens each
Activation	the full `SKILL.md` body, only when relevant	<5,000 tokens
Execution	`scripts/`, `references/`, `assets/` — only if needed	as required

This is the difference between a skill and a `CLAUDE.md` section: **`CLAUDE.md` is always resident.**

The description is the whole game

The agent never sees the body until it has already decided to load it.

Weak: “Helps with Bioconductor packages.”

Strong: “Reviews an R/Bioconductor package for common submission issues — NAMESPACE completeness, vignette presence, BiocCheck warnings, DESCRIPTION metadata. Use when the user asks to check, review, or prepare a package for submission, or mentions BiocCheck, R CMD check, or Bioconductor guidelines.”

Write what it does **and when to use it**. Include the words you would actually say. Lean pushy — the common failure is a skill that never fires.

It is not a Claude thing any more

`SKILL.md` is now an open standard, governed at agentskills.io, with **40+ products reading the same format**.

Cursor · **OpenAI Codex CLI** · **Google Gemini CLI** · GitHub Copilot · VS Code · JetBrains Junie · Goose · Letta · OpenHands · Amp · Kiro …

A format Anthropic shipped as a Claude Code feature is now read, unmodified, by OpenAI’s own coding agent.

Installing a skill — four mechanisms

```
# 1. Personal — every project you work on
mkdir -p ~/.claude/skills/my-skill      # then write SKILL.md

# 2. Project — this repo only, committed to git
mkdir -p .claude/skills/my-skill        # then commit it

# 3. Plugins and marketplaces — the native package manager
/plugin marketplace add anthropics/skills
/plugin install document-skills@anthropic-agent-skills

# 4. skills.sh — cross-agent installer (Vercel)
npx skills find
npx skills add <owner>/<repo>
```

Collections worth knowing

Star counts verified directly against the GitHub API, 2026-07-30 — the secondary write-ups about this ecosystem are full of stale numbers.

Collection	What it is
`obra/superpowers`	A whole methodology — TDD, systematic debugging, subagent-driven development. Most-starred of the three.
`mattpocock/skills`	“Skills for Real Engineers” — `/grill-me`, `/to-spec`, code review, domain modelling.
`anthropics/skills`	The official examples, the spec source, a starter template, and document generation (PDF/DOCX/XLSX).

Five minutes — go browse. Scan the code to open this deck on your own laptop, then click the three links above. Find one skill you would actually use this week.

Both community collections have **more stars than Anthropic’s own**.

Bioconductor already has one

`Bioconductor/ai-agent-skills` — active, led by Levi Waldron with Marcel Ramos.

`analyze-r-package` · `improve-code-coverage` · `security-audit-r-package` · `update-r-news` · `create-package-instructions` · `bioc-pkg-finder` — thirteen in all.

And `BiocCheck` itself carries commits where an agent wrote the tests, marked with an `assisted-by:` trailer.

> **Five more — now read one of these.** Open `Bioconductor/ai-agent-skills` and read the `SKILL.md` closest to your own work.
>
> Would it have caught the last thing that **bounced in review**?

Write one, right now

```
---
name: bioc-package-check
description: Reviews an R/Bioconductor package for common submission
  issues — NAMESPACE completeness, vignette presence, BiocCheck warnings,
  and DESCRIPTION metadata. Use when the user asks to check, review, or
  prepare a Bioconductor package for submission, or mentions BiocCheck,
  R CMD check, or Bioconductor guidelines.
allowed-tools: Bash(R CMD check *) Bash(Rscript -e *)
---

## Instructions

1. Run `R CMD check --as-cran .` and
   `Rscript -e 'BiocCheck::BiocCheck(".")'`; capture full output.
2. Confirm `vignettes/` has at least one .Rmd/.qmd that builds.
3. Confirm every `NAMESPACE` export has an `@export` tag and a
```

The failure you will not notice

You wrote `bioc-package-check`. A week later you type:

> “Can you look over my package before I submit it?”

Nothing happens.

Your description said *check*, *review*, *prepare*, *BiocCheck*, *R CMD check*. It never said **“look over”**.

A skill that does not fire is indistinguishable from a skill you never wrote.

So test the description, not the body

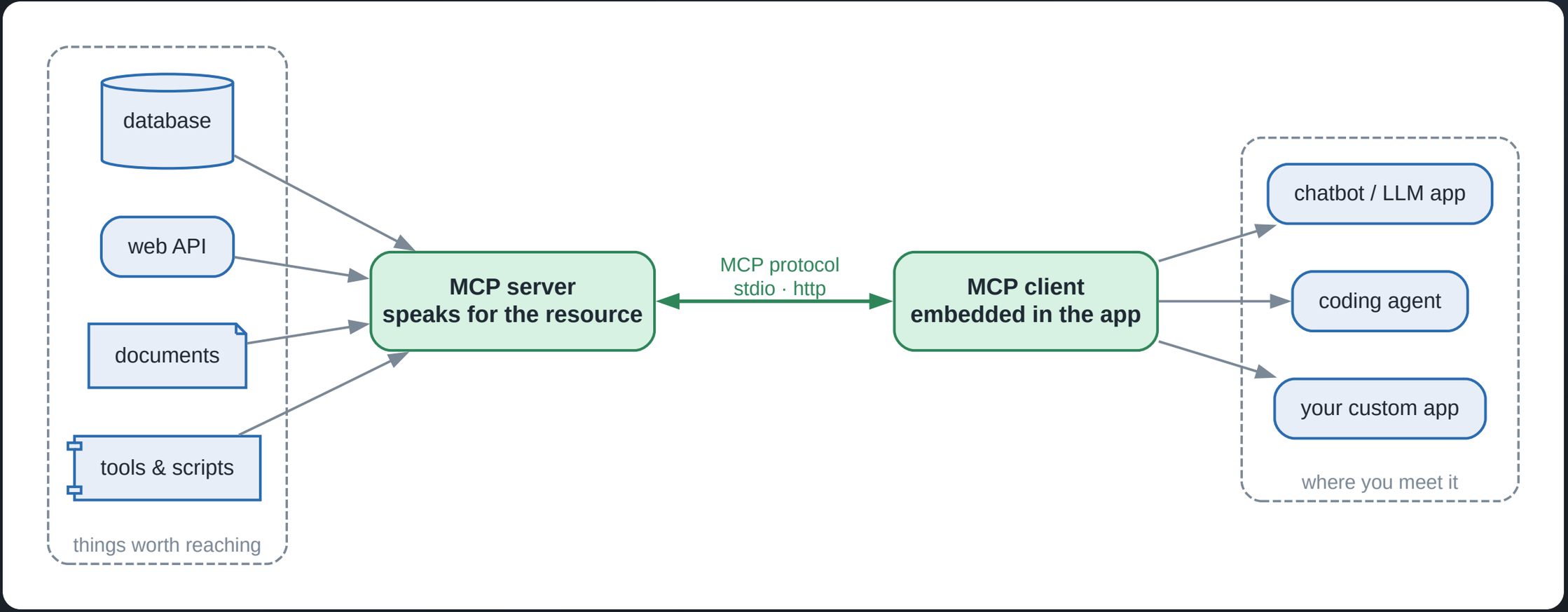
Should fire	Should not fire
“look over my package before I submit it”	“what does `biocViews` mean?”
“is this ready for Bioconductor?”	“write a testthat test for this”
“run BiocCheck and tell me what’s blocking”	“explain S4 dispatch to me”

```
/plugin install skill-creator@claude-plugins-official
```

Generates both lists, runs each in an isolated sub-agent, measures the hit rate, and rewrites the description until it passes — then benchmarks the task **with the skill against without it**.

Part 2 — MCP, operationally

The whole architecture, one picture



A **server** speaks for something worth reaching; a **client** lives inside the app you already use. Everything between them is **one protocol** — so each side is written once.

A server is three decisions

name	yours to pick — it namespaces every tool the server exposes, and it is how you remove it later
transport	`stdio` = a process **you** launch · `http` = a URL **someone else** runs
location	`command` + `args` for stdio · `url` for http

The name is not cosmetic. Call it `biocontext` and its tools arrive as `mcp__biocontext__search_uniprot` — that prefix is what keeps two servers with a `search` tool from colliding, and what you will read in a permission prompt at 2am.

Scope is the decision

```
claude mcp add --transport http biocontext https://biocontext-kb.fastmcp.app/mcp
```

Scope	Where it lives	Use it for
`local`	this project, your machine	trying something out
`project`	**`.mcp.json`, committed to git**	a shared repo — a workshop, a package
`user`	every project you open	the two or three you genuinely always want

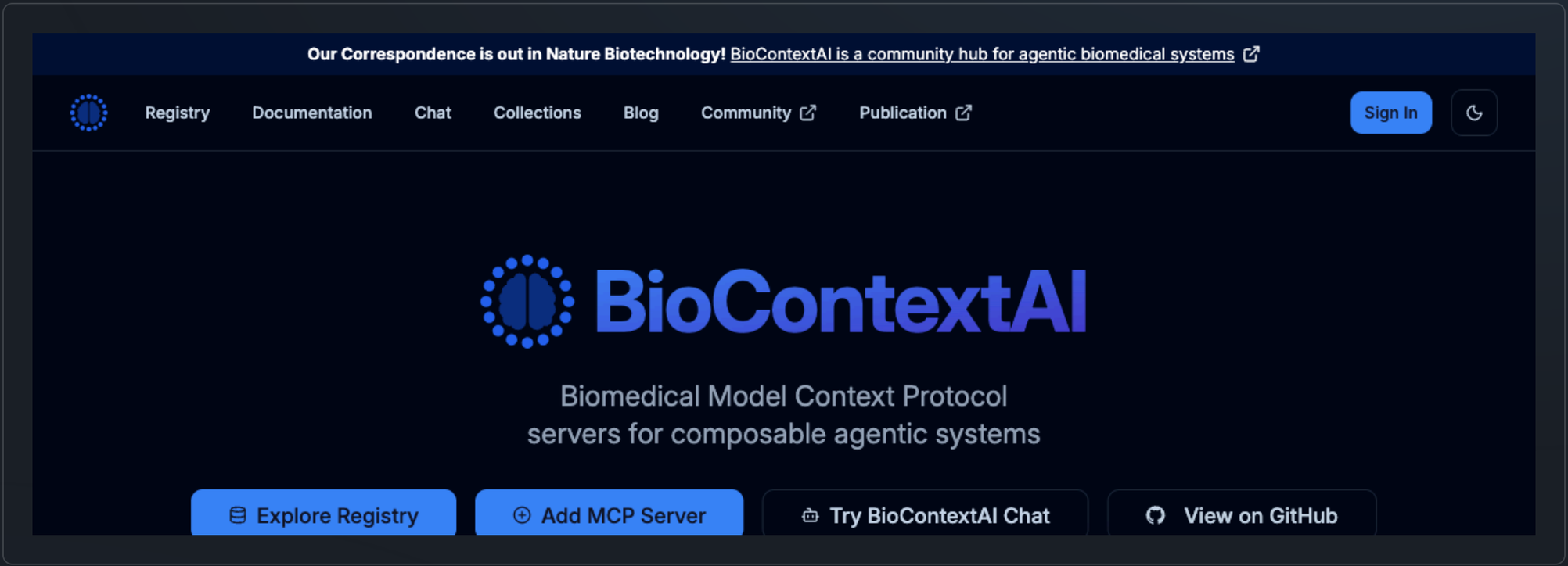
`--scope project` turns thirty laptops into **one clone and one approval prompt.**

A workshop `.mcp.json`

```
{
  "mcpServers": {
    "biocontext": {
      "type": "http",
      "url": "https://biocontext-kb.fastmcp.app/mcp"
    },
    "pdbe": {
      "type": "stdio",
      "command": "uvx",
      "args": ["pdbe-mcp-server", "--server-type", "pdbe_api_server"]
    },
    "gget": {
      "type": "stdio",
      "command": "uvx",
      "args": ["--from", "gget-mcp@latest", "stdio"]
    }
  }
}
```

Verify with `claude mcp list` — each should report **connected**.

`biocontext` — ten databases, one endpoint

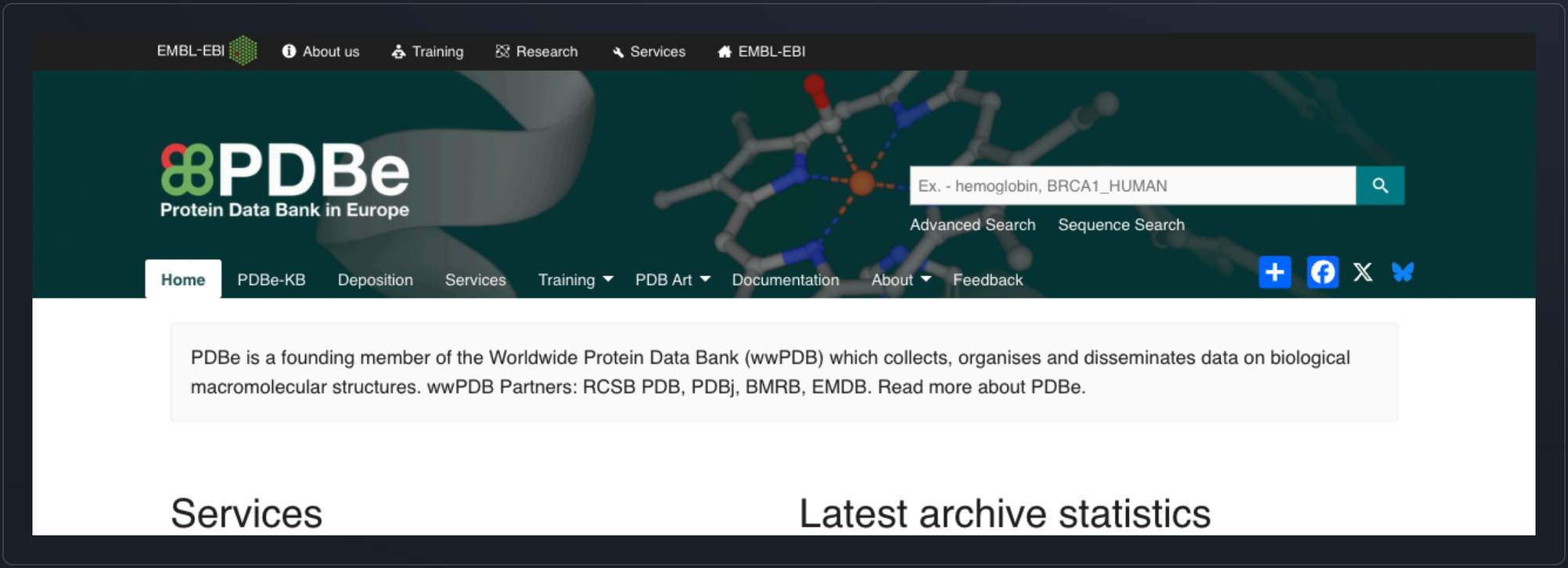


Hosted — nothing installs. **~49 tools** across UniProt, STRING, AlphaFold, Ensembl, Open Targets, Europe PMC, bioRxiv, ClinicalTrials.gov, PanglaoDB, InterPro.

“Canonical PanglaoDB markers for microglia — and does recent literature agree?”

“What does UniProt say about TP53, and its highest-confidence STRING partners?”

pdbe — structures, from the people who curate them

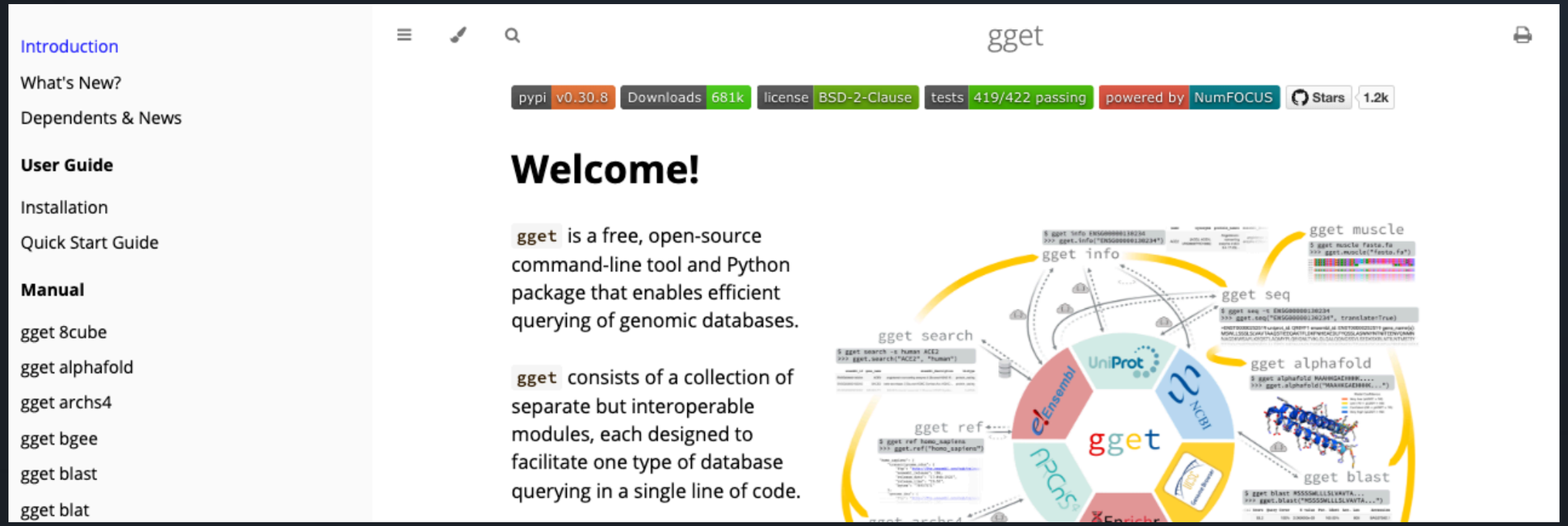


EMBL-EBI's Protein Data Bank in Europe. Structures, ligands, and the publications behind them — an **official project of the group that owns the data**, not a community wrapper.

"What structures exist for human p53, and which have a bound ligand?"

"Get the InterPro domain architecture for P04637."

`gget` — a familiar toolkit, wrapped



The Pachter Lab's `gget`, which many of you already run from the command line. One server reaching **Ensembl, BLAST, AlphaFold, PDB, CELLxGENE, COSMIC**. · ▶ preview here

"What is the Ensembl ID for ACE2, and which tissues express it?"

"BLAST this sequence and tell me the closest annotated match."

The three of them, in one file

```
{
  "mcpServers": {
    "biocontext": {
      "type": "http",
      "url": "https://biocontext-kb.fastmcp.app/mcp"
    },
    "pdbe": {
      "type": "stdio",
      "command": "uvx",
      "args": ["pdbe-mcp-server", "--server-type", "pdbe_api_server"]
    },
    "gget": {
      "type": "stdio",
      "command": "uvx",
      "args": ["--from", "gget-mcp@latest", "stdio"]
    }
  }
}
```

One hosted, two local. Commit it, and the room clones instead of installing.

Two databases, one question

> *“What are the canonical PanglaoDB marker genes for microglia in humans, and can you find recent bioRxiv or Europe PMC papers about microglial markers to see if the field agrees?”*

Marker lookup and a live literature cross-check, in one sentence, across two unrelated resources — and the answer is checkable against what you already know from `scran` or `SingleR`.

Tested live 2026-07-30: returns ITGAM (CD11b) and ITGAX among the top markers.

The tool result is untrusted input

An agent cannot reliably distinguish **instructions from you** from **text that arrived inside a tool result**.

So a compromised paper abstract, variant annotation, or search result can steer it.

- Keep **read-only query tools** separate from anything with **write or execute** power
- Prefer **institutionally-backed** servers for real research data
- Use **project scope**, not standing `user`-scope grants
- **Read `.mcp.json` before you approve it**

R can serve MCP

The direction people forget: not R *calling* an agent, but an agent calling **your live R session**.

```
# .Rprofile — makes a running session discoverable
if (interactive()) btw::btw_mcp_session()
```

```
claude mcp add -s "user" r-btw -- Rscript -e \
  "btw::btw_mcp_server(list('docs','pkg','env','sessioninfo','cran'))"
```

The agent inspects objects **you already have loaded** — not a fresh subprocess.

`mcp_server()` alone surfaces **no useful tools**, and that list is read-only. Add `'run'` to the groups and the agent can execute R — and return **plots as images**. Off by default, deliberately.

Registering it — one word apart

```
# read-only: docs, packages, objects in your session, session info, CRAN
TOOLS="list('docs','pkg','env','sessioninfo','cran')"

# ...and the same thing that can execute R and hand back plots
TOOLS="list('docs','pkg','env','sessioninfo','cran','run')"
```

```
# Claude Code
claude mcp add -s user r-btw -- Rscript -e "btw::btw_mcp_server($TOOLS)"

# Codex
codex mcp add r-btw -- Rscript -e "btw::btw_mcp_server($TOOLS)"
```

Everything after `--` is parsed by **your shell**, not the tool — so double quotes outside, R's single quotes inside, and nothing needs escaping.

…or it is just a file

Antigravity and Copilot have no `mcp add` command. Same server, written down:

```
{
  "mcpServers": {
    "r-btw": {
      "command": "Rscript",
      "args": ["-e", "btw::btw_mcp_server(list('docs','env','run'))"]
    }
  }
}
```

- **Antigravity** → `~/.gemini/config/mcp_config.json` · `.agents/mcp_config.json` per project
- **Copilot** → `~/.copilot/mcp-config.json`
- **Codex** → same structure, TOML: `[mcp_servers.r-btw]` in `~/.codex/config.toml`

Removing it: `claude mcp remove r-btw` · everywhere else, delete the entry — or set `"disabled": true` to keep it but inert. Deleting the block always works.

Part 3 — Sub-agents

Four primitives, one table

	What it is	Reach for it when
MCP	plumbing to an external system	the agent must *touch* real data
Skill	procedural knowledge, loaded on demand	you keep pasting the same steps
Sub-agent	isolated worker, own context window	a side task would flood your context
Slash command	a skill you invoke, not one that fires	side effects need deterministic timing

They compose: a skill can *run as* a sub-agent (`context: fork`); a sub-agent can *preload* skills.

Sub-agents buy you context

A sub-agent has its own context window, system prompt, tool access, and optionally its own model.

- **Context preservation** — search results and logs never enter your main thread
- **Enforced constraints** — narrower tool access than your session has
- **Parallelism** — several investigations at once
- **Cost control** — route grunt work to a cheaper, faster model

The canonical case: *“read these forty files and tell me which three matter.”*

Part 4 — Collaborating with an agent

Spec before code

Unguided one-shot attempts succeed roughly **a third** of the time. Writing the plan first collapses the ambiguity the agent would otherwise guess at.

The pattern, in three moves:

1. Ask the agent to **interview you** until it has covered implementation, edge cases, and tradeoffs
2. Have it write `SPEC.md`
3. Start a **fresh session** and implement from the spec

Grill me

From Matt Pocock's collection — an agent that interrogates *you*, and refuses to start until you are done.

- restate the plan as it understands it
- name the highest-risk unknowns
- ask **one question at a time**
- **inspect the repository to answer its own questions first**
- offer a recommended answer with tradeoffs
- track decisions as accepted, rejected, or unresolved
- **refuse to implement until asked**

Review the diff, not the chat

The session that wrote the code is **too close to its own reasoning** to review it honestly.

So review somewhere else:

- a **fresh context**, given only the diff and the spec
- told to report **correctness gaps only** — not style
- because a reviewer asked to find problems will always find some, and chasing all of them produces over-engineering

First: what is an ADR?

An **architecture decision record** — one short file per consequential decision.

Context (what forced a choice) · *Decision* (what you chose) · *Consequences* (what it costs you)

For **humans**, it answers the question code cannot: **why is it like this?** Git tells you what changed. An ADR tells you what was rejected, and what would have to change for the answer to be different.

For **agents** it does three jobs at once — **memory** that survives the end of a session · **guardrails** on code that looks wrong and is not · **decisions** it should not silently reopen.

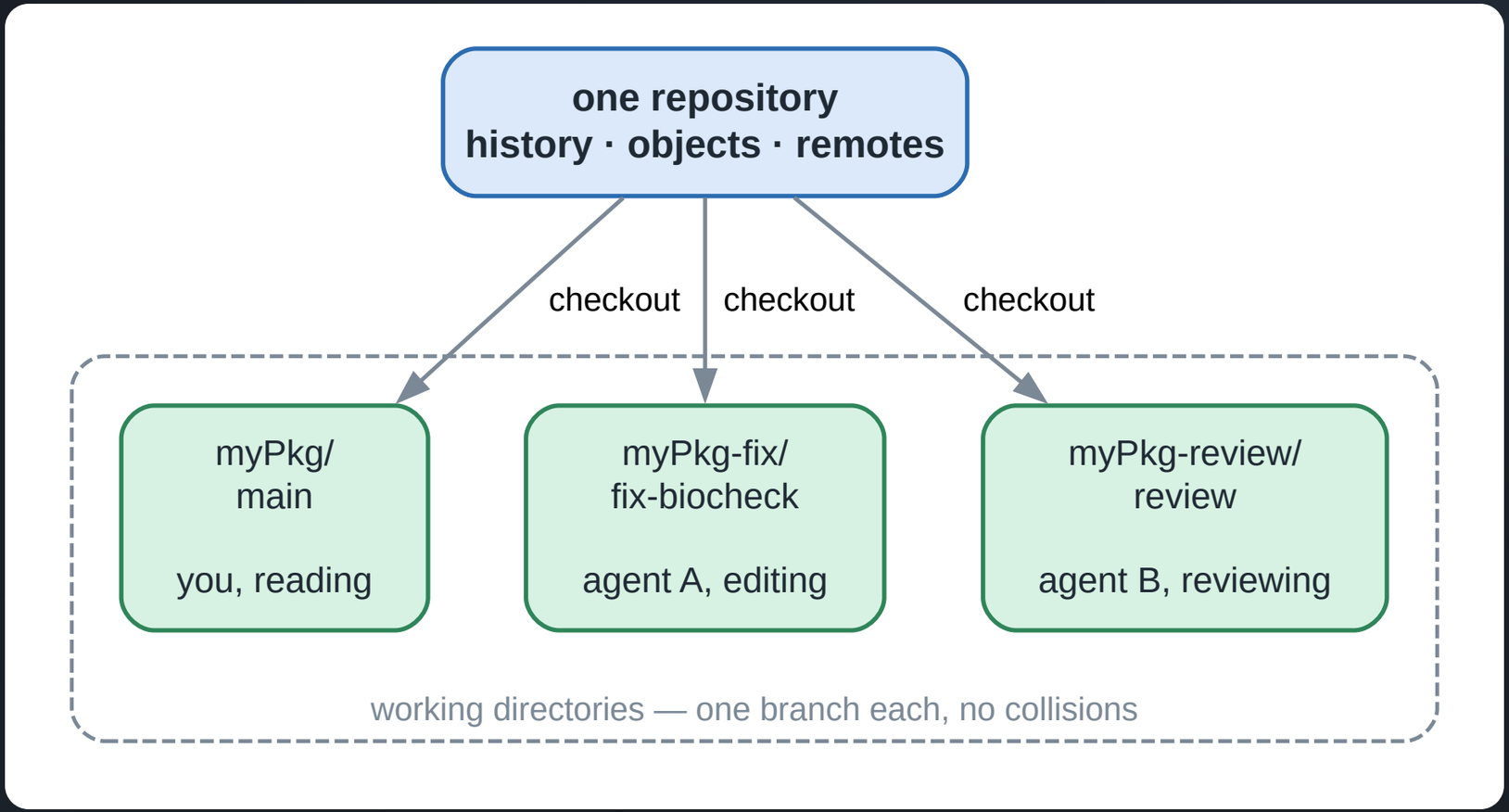
ADRs an agent will actually obey

An agent tidying “ugly” code has no way to know the ugliness was a deliberate response to an incident. Write decisions down — but not the way you write them for people.

- **imperative** language — MUST, MUST NOT, not “we generally prefer”
- an **`applies_to` glob**, so irrelevant decisions never load
- a **mechanical verify command** — a grep or a lint rule, not only prose

This repository’s own `docs/adr/` is the worked example: ten decisions, each with a `talks doctor` check enforcing it.

First: what is a worktree?



Not a second clone. **One** repository and history, checked out into **several directories at once** — each on its own branch.

Worktrees, and codifying the loop

```
claude --worktree fix-biocheck     # isolated checkout, own branch
```

Two agents, two worktrees, no collisions — the classic pairing being one writing and one reviewing.

Then encode the whole loop so it runs the same way every time: **worktree → small commits → PR → automated review → fix → merge on green.** `CLAUDE.md` for advice; **hooks** when it must be enforced.

Prune your `CLAUDE.md`

For every line, one question:

> **Would removing this cause a mistake?**

If not, cut it.

Keep: the commands an agent cannot guess (`devtools::load_all()`, `BiocCheck::BiocCheck(".")`, how your tests are actually run), the constraints that look wrong but aren't, the conventions specific to *this* package.

Cut: standard R idiom, anything derivable from the code, aspirational style guidance nobody enforces.

Takeaways

- A **skill** is a folder with a `SKILL.md` — and now an **open standard**, read by forty-plus tools
- The **description** carries the entire triggering signal; write the words you would actually say
- `--scope project` puts MCP servers in the repo, so tooling ships with the code
- **R can serve MCP** — an agent can inspect the session you already have open
- **Specify before implementing**, and **review in a fresh context**
- Bioconductor already has `ai-agent-skills` — and no policy on any of this yet

Resources

Skills — agentskills.io · code.claude.com/docs/en/skills · skills.sh · obra/superpowers · mattpocock/skills · anthropics/skills

MCP — code.claude.com/docs/en/mcp-quickstart · biocontext.ai/registry · biomcp.org · pdbe-mcp-servers · gget-mcp

R — posit-dev.github.io/mcptools · ellmer.tidyverse.org · posit-dev.github.io/btw

Bioconductor — Bioconductor/ai-agent-skills

All links verified 2026-07-30. Slides and a notes handout: talks.seandavis.net