06: Harnesses, tools, and MCP

Agent = model + harness

Why this matters

- The same model can feel brilliant in one harness and clumsy in another.
- "The model got worse" is often a harness change.
- Tools and permissions decide what an agent can safely do.
- The same concepts apply across products.

Agent = model + harness

```
Harness = agent loop + tools + permissions + context management + UI
Agent   = model + harness
```

```
your task → model → tool call ("run the tests")
          → harness checks permission, runs it
          → result goes into the context → model → ...
          → model says done → answer, diff, or PR
```

The model is the engine. The harness is the rest of the car.

"Claude Code got worse." The weights hadn't changed.

Anthropic's April 2026 postmortem traced user complaints to three harness changes:

- A lower default reasoning effort
- A bug that cleared earlier reasoning on every turn
- A system-prompt line capping text between tool calls

All three were reverted or fixed. Evaluate a model in the harness you'll actually use.

Part 1

From autocomplete to orchestrators

Six generations, all still in use

#	Generation	Arrived	What you do
1	Autocomplete	2021	Type; it suggests the next line or block
2	Chat	late 2022	Paste code in, copy answers back
3	IDE agents	2023–2025	Describe a change; accept its diffs
4	CLI agents	2025	Give a task in the terminal
5	Cloud or background agents	2024–2025	Assign a task, walk away, get a PR
6	Desktop GUIs and orchestrators	2025–2026	Supervise several agents at once

Why the terminal won, and what wrapped it

- The shell is the universal tool: build, test, lint, query, `gh`.
- An agent that runs commands needs no per-service integration.
- CLI agents run headless: in CI, in scripts, in cloud sandboxes.
- Generation 6 wraps generation 4: GUIs that drive your installed CLIs.
- They add parallel sessions, one git worktree each, side-by-side diffs.

The market churns (as of November 2026)

- Gemini CLI's individual users moved to Antigravity CLI, June 18, 2026
- SpaceX acquired Cursor, August 14, 2026
- Windsurf split in July 2025; Google hired its leaders, Cognition bought the rest
- Roo Code's extension shut down May 15, 2026
- Continue is archived and read-only
- Amp spun out of Sourcegraph, late 2025

Check before you standardize on anything.

Part 2

Tools and MCP

A tool is a name, a description, and a schema

```
{
  "name": "run_tests",
  "description": "Run the test suite. Use after editing code.",
  "input_schema": {
    "type": "object",
    "properties": { "path": { "type": "string" } }
  }
}
```

- Descriptions are prompts. Vague ones cause wrong tool choices.
- Every definition costs context, on every model call.

MCP is one protocol for tools and data

- Without it, 10 harnesses × 100 services = 1,000 custom integrations.
- With MCP, 10 clients plus 100 servers.

When	What happened
Nov 25, 2024	Anthropic releases the Model Context Protocol
Mar–May 2025	OpenAI, Google, and Microsoft adopt it
Dec 9, 2025	Donated to the Linux Foundation's Agentic AI Foundation
Jan 26, 2026	MCP Apps lets tools return interactive UI
Jul 28, 2026	A stateless spec: no handshake, no sessions

Host, client, server

Term	Meaning
Host	The AI app you use: Claude Code, Codex, Cursor, VS Code, ChatGPT
Client	The connector inside the host that talks to one server
Server	A program exposing capabilities: GitHub, Postgres, a browser
Tools	Functions the model can call
Resources	Data the app can read, addressed by URI
Prompts	Reusable templates, often shown as slash commands
Transports	stdio for local subprocesses; Streamable HTTP for remote servers, usually with OAuth

To the model, an MCP tool looks like any other tool.

MCP's costs are real

- Context bloat, with "58 tools consuming approximately 55K tokens" up front
- A server is code running with your credentials
- Servers that read web pages, issues, or email are injection channels
- Quality is uneven; reference servers are "not production-ready solutions"
- Or just use the CLI; agents already know `gh`, `psql`, `kubectl`

Three ways to give an agent a capability

Option	Strengths	Costs
MCP tool	Structured, discoverable, OAuth; works without a shell	Context; uneven quality
CLI in a shell	Known from training; composes with pipes; free until used	Needs a shell and a sandbox; no standard auth
Code mode	One Anthropic example: 150,000 → 2,000 tokens	Needs a secure execution environment

Use MCP for integrations others will use. For your own agent, try the CLI first.

Part 3

The shared toolkit

The kit every major harness converged on

Feature	Use it for	Watch out for
Instruction files	Commands, conventions, gotchas	Bloat (module 07)
Skills	Repeatable procedures	They can run code
Subagents	Exploring, reviewing, noisy work	Extra tokens; lossy summaries
Hooks	Deterministic enforcement	A buggy hook blocks all work
Plugins	Sharing a team's setup in one install	Supply-chain risk
Permission modes	Matching autonomy to risk	"Bypass all" on your real machine
Sandboxing	Longer runs without approvals	Leaky allowlists
Git worktrees	Parallel agents without collisions	Disk use; clean up after
Checkpoints	Undoing a bad turn	Not a replacement for commits

Skills load on demand. Hooks always run.

	Skills	Hooks
What	A folder with `SKILL.md` and optional scripts	Your shell commands at lifecycle events
Loads	Description up front; body only when relevant	Every time the event fires
Who decides	The model, when a task calls for it	No one: the harness always runs it
Use for	Releases, migrations, other procedures	Format on save; block `rm -rf`; no "done" while CI is red

Prose in an instruction file is a request. A hook is a rule.

Approval prompts stop working when you always say yes

- Users approve 93% of permission prompts (Anthropic, March 2026).
- In auto mode, a classifier approves routine actions; hard blocks stay.
- It still missed 17% of "overeager" actions in Anthropic's tests.
- Long unattended runs need a real sandbox with limited credentials.

Closed bundles put model, harness, and tools in one

Product	Good for
Claude Design (Apr 2026, research preview)	Idea to clickable prototype or deck
Claude in Chrome	Web workflows; testing web apps
Codex app and automations (Feb 2026)	Supervising parallel tasks; recurring triage
ChatGPT agent and Atlas	Research and multi-step web tasks
Cursor with Composer	Fast in-editor agent loops
Google Antigravity	Teams centered on Gemini models
Devin, Jules, Copilot coding agent	Well-scoped backlog tickets

Convenient, but you can't see inside, swap parts, or stop them changing.

How to choose a harness

Question	Look for
Model access	One vendor's models, or bring any model?
Cost model	Subscription, per-token API, or seats (module 12)
Team standards	Reads `AGENTS.md`? Supports Agent Skills and MCP?
Privacy	Retention terms, zero data retention, local or self-hosted
Extensibility	Hooks, skills, plugins, MCP, an SDK
Autonomy and safety	Permission modes, sandboxing, worktrees, audit logs
Durability	Open license, foundation governance, portable config

Don't choose on demo videos.

A sane baseline setup

1. Pick one CLI agent. Learn its permission modes first.
2. Write a short `AGENTS.md` with commands and non-obvious conventions.
3. Add hooks for whatever must always happen.
4. Connect only the one to three MCP servers you need.
5. Use worktrees before running two agents on one repo.
6. Sandbox before you turn off approvals. Commit often.

Bryan's take

Bryan's take: My daily setup is several CLI agents running in parallel, each in its own worktree, which I manage by hand. I drive them through T3 Code, which lets me run Claude and Codex side by side, the two model families I think are best for code (module 03). That puts me at roughly stage 6–7 on Yegge's eight-stage ladder (module 08). I'm not running a fully autonomous orchestrator, because I haven't gotten reliable results from one.

Try it

Individual (60–90 minutes)

- Install a CLI agent; note the context used before you type.
- Connect one MCP server and measure the tokens it adds.
- Add a formatter hook, then a Stop hook that waits for green tests.

Class or team (one session)

- Same small task, same model tier, three different harnesses.
- Record time to green, interventions, cost, and diff quality.
- Discuss which differences came from the harness, and which from people.

Discussion

1. Agent = model + harness. Which part matters more for your work today, and why?
2. When should a capability be an MCP server, a CLI, or a skill with a script?
3. Prompts get approved about 93% of the time. Is an always-approved prompt a safety feature?
4. Tools were acquired, pivoted, or shut down in 2026. How does a team protect itself?

Read more

- Module: 06 Harnesses, tools, and MCP
- MCP documentation
- Code execution with MCP (Anthropic, November 2025)
- April 23 postmortem (Anthropic, April 2026)