XXIII

Jak dnes ventiluješ nápad do hotové věci?

Po této kapitole budeš znát dílnu roku 2026 — terminálové agenty i editory s agentem uvnitř — a hlavně budeš vědět, kdy sáhnout po kterém. A budeš mít detektor, kterým každý nový nástroj proženeš dřív, než mu uvěříš: přidává řešiteli sílu, nebo jen leští povrch?

> *Je to nový druh programování, kterému říkám „vibe coding“ — kdy se úplně poddáš vibru, přijmeš exponenciály a zapomeneš, že kód vůbec existuje.*
>
> — Andrej Karpathy, *X (dříve Twitter), 2. února 2025 — „…fully give in to the vibes, embrace exponentials, and forget that the code even exists.“ O rok později (únor 2026) autor termín rozdělil: „vibe coding“ nechal experimentálnímu kódu, produkci pokřtil „agentic engineering“.*

Otevřu terminál. Ne editor, ne okno s tlačítky — obyčejné černé okno, do kterého se píše. Napíšu jediné slovo:

```
claude
```

Objeví se řádek, který čeká. Napíšu do něj česky, jako bych to říkal kolegovi přes stůl: *„Postav mi jednoduchou webovou hru — pexeso s osmi páry karet, skóre, tlačítko Nová hra. Jeden soubor HTML, ať to jde poslat mailem.“* A zmáčknu Enter.

Nestane se zázrak, stane se něco lepšího, protože je to viditelné. Nahoře naskočí věta *„Přemýšlím…“*, pak seznam kroků, které agent hodlá udělat. Vzápětí vidím, jak *sahá po nástroji*: `Write karty.html`. Pod tím vyskočí rozdíl — zelené řádky, které přibyly, kus po kuse, jak píše. Za pár desítek vteřin řekne „hotovo“ a vypíše, jak to spustit. Otevřu soubor v prohlížeči. Karty se otáčejí, skóre naskakuje, tlačítko funguje. Nenapsal jsem ani jednu závorku. Neznám nazpaměť jediný atribut, kterým se v HTML nastaví mřížka. A přesto hra běží.

> Pamatuju si, jak jsem první takovou věc postavil, a stydím se za tu emoci: byl to *úlek*. Ne radost, úlek. Dvacet let jsem platil za každý

> řádek nocí — a tady mi za minutu vzniklo něco, co bych kdysi stavěl celý večer. První reakce nebyla „skvělé“, byla „*tak k čemu potom já?*“. Trvalo mi pár dní, než jsem otočil otázku správně: ne *k čemu já*, ale *co teď dělám já, když syntaxe už není moje práce*. Zbytek téhle knihy je odpověď.

To je celá scéna, kolem které se točí tahle kapitola. Nápad → věta → nástroje → rozdíl → hotová věc. Chci ti ukázat, co se v tom černém okně doopravdy děje, jaké dílenské stroje na tenhle pohyb dnes existují, a hlavně kdy sáhnout po kterém. Protože nástrojů je víc a marketing každého křičí, že je „revoluční“ — a přesně na tenhle křik máme z druhé kapitoly detektor.

Verze k červenci 2026 (ověřeno oproti oficiálním docs a release notes). Nástroje se mění po týdnech — čti čísla verzí jako fotografii*, ne jako věčnou pravdu. Metoda pod nimi je stálá; to je celý smysl dělení knihy na dvě části.*

Anatomie jednoho sezení

TEENAGER · MECHANISMUS

Ať sedíš u kteréhokoli z dnešních agentů, ten pohyb v okně má vždycky tutéž kostru. Pět kroků, dokola:

```
# 1) PROMPT  — položíš úlohu vlastními slovy na
pracovní stůl
„Postav pexeso s osmi páry, skóre a tlačítkem
Nová hra.“

# 2) PLÁN    — agent rozloží úlohu na kroky (a
smíš je schválit)
→ vytvořit karty.html · mřížka 4×4 · logika párů
· skóre

# 3) NÁSTROJE — agent SÁHNE po nářadí: čte, píše,
spouští
Read  index.html
Write karty.html
Bash  „otevři v prohlížeči a zkontroluj konzoli“

# 4) DIFF    — ukáže rozdíl: co přibylo, co
ubylo, řádek po řádku
+ <div class="karta" ...>           (zelené =
přidané)
- <!-- placeholder -->              (červené =
smazané)

# 5) COMMIT  — když to sedí, uložíš pozici do
gitu
git commit -m „pexeso: základní hra“
```

Kroky 1 a 5 jsou tvoje — položit úlohu a rozhodnout, že výsledek je dost dobrý na uložení. Kroky uprostřed dělá stroj. A tady je celý fígl éry: to, co kdysi bývalo *tvou* prací (napsat každý řádek), se přesunulo dovnitř kroku 3; to, co zůstalo tobě, je *úsudek* na koncích — dobře položit úlohu a poznat, že výstup je špatně. Přesně to je kentaur z první kapitoly, jen teď u klávesnice: člověk drží proces, stroj drží syntaxi.

Dvě slova z té kostry si zaslouží vysvětlení, protože se pletou i lidem, co je používají denně.

Prompt je text, který modelu položíš na pracovní stůl — vzpomeň na obraz z kapitoly 15: kontextové okno je pracovní stůl, ne knihovna. Model ví jen to, co mu na stole zrovna leží. Když mu neřekneš, že hra má fungovat i na mobilu, neví to; není líný, jen mu to nikdo nepoložil.

Diff (rozdíl) je pohled na to, *co přesně* se změnilo — zeleně přidané řádky, červeně smazané. Je to nejdůležitější okno celého sezení, protože

diff — z anglického difference. *Vývojářská čeština říká „dif“ nebo „rozdíl“; sloveso „udělat diff“ je legální. Vrací se v kap. 26: PR review je čtení diffu cizí (i strojové) práce před sloučením.*

je to místo, kde přestáváš věřit a začínáš ověřovat. Agent ti neukazuje jen výsledek („hotovo“); ukazuje ti *svůj tah*, dřív než mu uvěříš. Kdo diff nečte, řídí se zavřenýma očima.

Dvě dílny: terminál a editor

TEENAGER · MECHANISMUS

Nástroje roku 2026 se dělí na dva rody podle toho, *kde* ten agent bydlí.

Terminálový agent žije v příkazové řádce. Napíšeš `claude` (nebo `codex`, nebo `agy`) a mluvíš s agentem textem v okně. Žádná tlačítka, žádné panely — jen konverzace a nářadí, po kterém agent sahá. Zástupci: **Claude Code** (Anthropic), **Codex CLI** (OpenAI), **Antigravity CLI** (Google), **OpenCode** (open-source, SST).

Editor s agentem uvnitř (IDE) je grafické prostředí — okno s panely, stromem souborů, zvýrazněnou syntaxí — do kterého někdo přišil agenta jako postranní panel. Zástupci: **Cursor**, **Antigravity IDE** (Google), **GitHub Copilot** ve VS Code. Tady vidíš kód i agenta najednou; klikáš i píšeš.

Hrubé pravidlo, které platí, dokud si nevytvoříš vlastní cit:

```
# KDY TERMINÁL (CLI)
#  – chceš skládat kroky za sebe (build → test
→ deploy)
#  – chceš agenta poslat na dávku úloh a nechat
běžet
#  – chceš to samé spustit zítra znovu jedním
řádkem
#  – pracuješ na serveru, kde žádné okno není

# KDY EDITOR (IDE)
#  – potřebuješ vidět velký soubor a klikat po
něm
#  – učíš se, chceš zvýraznění a doplňování pod
rukou
#  – laděním v malém, kde je vizuální kontext k
nezaplacení
```

Proč vůbec rozlišovat, když oba umějí totéž — postavit pexeso za minutu? Protože se liší v jedné věci, která je pro tuhle knihu klíčová, a dostane vlastní oddíl níž: v tom, jestli se dají *skládat*. Napřed ale dílny samy.

Terminál konkrétně: tři agenti a jejich povaha

TEENAGER · MECHANISMUS

Claude Code (Anthropic) — verze 2.1.211, model Opus 4.8 jako default na cloudových platformách, k 15. 7. 2026. Od dubna 2026 běží jako *nativní binárka*: startuje v desítkách milisekund, ne vteřin (bootstrap Node.js zmizel). Instalace jedním řádkem:

```
curl -fsSL https://claude.ai/install.sh | bash
# macOS / Linux
```

Čtyři věci, kterými se vymyká — a které rozebereme v dalších kapitolách:

```
# PAMĚŤ — čte hierarchii CLAUDE.md (specifický
přepíše obecný)
~/.claude/CLAUDE.md      # tvoje preference
napříč projekty
./CLAUDE.md              # konvence týmu
(COMMITni do gitu!)
./src/auth/CLAUDE.md     # specifika složky,
lazy-load při vstupu

# HOOKS — shell příkaz spuštěný při události
(pojistka, kap. 24)
# SKILLS — tvé řemeslo zabalené do SKILL.md (kap.
25)
# SUBAGENTS — delegace na samostatné agenty s
vlastním kontextem
# MCP — univerzální zásuvka na okolní svět (kap.
24)
```

TEENAGER · MECHANISMUS

Codex CLI (OpenAI) — přepsaný z TypeScriptu do Rustu, model řady GPT-5.6 (dostupný od července 2026), k 15. 7. 2026. Kde Anthropic sází na paměť a expresivitu, OpenAI svádí jiný boj: *bezpečnost při autonomním běhu*. Klíčový rozdíl je sandbox vynucený samotným operačním systémem — na macOS profily Seatbelt, na Linuxu Landlock + seccomp; síť je ve výchozím režimu *vypnutá*.

```
npm i -g @openai/codex
codex --sandbox workspace-write   # default:
čte+píše projekt, síť off
codex --sandbox read-only         # jen čte
codex --full-auto                 # workspace-
write + ptá se jen při úniku
```

Projektový kontext čte ze souboru `AGENTS.md` (stejná myšlenka jako `CLAUDE.md`). `codex /init` ho založí.

TEENAGER · MECHANISMUS

Antigravity CLI (Google) — psaný v Go, spouští se příkazem `agy`, k 15. 7. 2026. Vznikl z nutnosti: Google **18. června 2026 zrušil Gemini CLI** pro tiery free/Pro/Ultra a nahradil ho platformou Antigravity. CLI je její terminálová část; běží na modelech řady Gemini 3 (Gemini 3.5 Flash je dnes výchozí Flash model). Binárku stáhneš přímo, žádný `npm`.

```
agy                    # interaktivní session
agy auth login         # přihlášení přes
prohlížeč (OAuth)
@src/hra.js            # @-zmínka vloží soubor do
kontextu
/artifacts             # plán + walkthrough,
které agent průběžně píše
/diff                  # co agent změnil ve VCS,
tah po tahu
```

Projektový kontext: `AGENTS.md` (ne dřívější `GEMINI.md`); dovednosti se přestěhovaly do `.agents/skills/`.

Pozor na dvojznačnost jména „Antigravity“: znamená tři *různé věci — IDE (editor od Googlu), desktopovou platformu 2.0 a terminálové CLI. Když ti někdo řekne „použij Antigravity“, ptej se* které.

> *Ta zrušená štědrost stojí za jednu poznámku. Gemini CLI mělo tisíc requestů denně zdarma, bez kreditky — „nejštědřejší free tier v branži“ — a vydrželo přesně do chvíle, než si na něm lidé zvykli. Pak zmizelo a zůstal jen návyk. Když nástroj rozdává, ptej se, na co si tě zvyká; účet přijde, až budeš závislý.*

Čtvrtý terminálový agent, **OpenCode** (open-source, MIT, SST), stojí za zmínku pro jednu vlastnost: *přines si vlastní model*. Funguje se sedmdesáti a více poskytovateli — Anthropic, OpenAI, Google, ale i lokální Ollama. Nezávislost na jednom dodavateli za cenu práce s nastavením. K ekonomice téhle volby (a k tomu, kdy se vyplatí lokální model) se vrátíme v kapitole 30.

→ kap. 30: co to celé stojí. BYOM (bring your own model) je pojistka proti lock-inu *— uzamčení u jednoho dodavatele. Svoboda má cenu v nastavení; jestli se ti vyplatí, spočítáš, ne odhadneš.*

Editor konkrétně: když chceš vidět a klikat

O editorech stačí míň, protože princip znáš — je to okno s panely, jen do něj přibyl agent. **Cursor** je fork VS Code s agentem zabudovaným do jádra; **GitHub Copilot** žije jako rozšíření uvnitř VS Code; **Antigravity IDE** je Googlova varianta (public preview, ne finální vydání, k červenci 2026). Všechny umějí totéž co terminál — položíš úlohu, agent sáhne po nástrojích, ukáže diff — jen to vidíš v grafickém obalu a můžeš mezi tím klikat po kódu.

Kdy sáhnout po editoru: když se *učíš* a chceš mít kód pod očima se zvýrazněním a doplňováním; když laděním procházíš jeden velký soubor a vizuální kontext ti šetří hlavu; když ještě nemáš v prstech příkazovou řádku a černé okno tě děsí — což je legitimní a nikdo se za to nemá stydět. Editor je skvělý první domov. Ale je to domov s nižším stropem, a proč, to ukazuje další oddíl.

A tady dobrá zpráva, kvůli které se nemá cenu bát: mezi terminálem a editorem se dnes *nemusíš rozhodnout napořád*. Kontext žije v obyčejném textovém souboru (`CLAUDE.md`, `AGENTS.md`), git je společný pro obojí a MCP servery fungují napříč nástroji. Postavíš pexeso v editoru dopoledne a odpoledne pošleš terminálového agenta, ať k němu dopíše deset her — a ani jeden se nediví. Volíš nástroj k úloze, ne úlohu k nástroji.

DEMO

`lesk-a-bida-vibecodingu.gaussalgo.com/d/dva-svety`

Postav tutéž drobnou hru dvakrát: jednou naklikáním v editoru, jednou jednou větou do terminálu — a změř, kolik z toho pohybu je *tvůj úsudek* a kolik jen syntaxe, kterou dnes napíše stroj.

Proč terminál, když chceš skládat

Teď to nejdůležitější, a je to přímé pokračování sedmnácté kapitoly. Vzpomeň na Excel noc a na `najdi | seřaď | spočítej`. Tam byla lekce jediná: *roura se skládá, klikání ne* — protože všechny příkazy

mluví stejnou řečí (text) a dají se navléknout za sebe jako korálky. Tatáž lekce platí o dílnách výš, jen místo tří příkazů skládáš celé kroky práce.

Terminálový agent totiž není jen chatovací okno. Je to *další článek do roury*. Podívej, co jde napsat:

```
claude -p „přidej test na výpočet skóre a spusť ho“
&& git commit -am „test skóre“
```

`-p` *(print) — neinteraktivní režim: jeden prompt, jeden výstup, žádná konverzace. Přesně ten tvar, který se dá zařadit do roury a spustit strojem, ne rukou. Ostatní CLI agenti mají obdobu.*

Dvě věci za &&: pošli agentovi úlohu neinteraktivně (`-p` = jeden prompt, vypiš výsledek) a když uspěje, rovnou commitni. Tohle je roura — výstup jednoho kroku spouští druhý. A protože je to text v souboru, spustíš to zítra znovu, dáš to do CI linky (kapitola 21), pošleš agenta na *dávku* úloh přes noc. Kompozice roste přidáváním článků.

Grafické rozhraní tohle *neumí*, a ne kvůli lenosti autorů. Umí to ze stejného důvodu, proč se neskládá klikání v Excelu: klik na tlačítko „Generuj“ v editoru nemá *výstup*, který bys předal dalšímu kroku. Editorový agent je konec cesty — dostaneš výsledek do okna a hotovo. Terminálový agent je *článek* — jeho výstup je vstup dalšího článku. To je rozdíl v druhu, ne ve stupni; přesně jako mezi nocí a vteřinou v sedmnáctce.

EINSTEIN · MATEMATIKA — přeskočitelné

Řekněme to formálně, protože je to tatáž algebra jako u rour. Terminálový agent v neinteraktivním režimu je *funkce*: vezme text (prompt, stav repozitáře) a vrátí text (diff, kód konce, log). Protože vstup i výstup jsou týž typ — text a soubory —, dá se složit s čímkoli dalším, co bere a vrací text:

$$(\text{commit} \circ \text{agent} \circ \text{test})(\text{úloha}) = \text{commit}(\text{agent}(\text{test}(\text{úloha}))).$$

Tahle skladatelnost je přesně to, co grafickému agentovi chybí: „klik na tlačítko“ nemá typ vstupu ani výstupu, protože žádný explicitní výstup neexistuje (kap. 17). Bez společného typu není skládání — a bez skládání každou úlohu spouštíš ručně, znovu a znovu. Editor tě vede za ruku; terminál ti dá páku, kterou zvedneš deset úloh najednou. Cena za páku je nižší pohodlí na první pohled — a přesně proto ji tolik lidí mine, stejně jako minuli rouru.

Neplyne z toho „terminál vždycky“. Plyne z toho: terminál, když chceš skládat a opakovat*; editor, když chceš vidět a klikat v jednom velkém souboru. Slovo „vždycky“ je zrádné z obou stran — snob i pohodlný se pletou stejně (kap. 17).*

Prožen to detektorem

A teď refrén, který jsme dostali ve druhé kapitole a slíbili nosit celou knihu. Každý z těch nástrojů má marketing, který o něm křičí, že je „revoluční“, „průlomový“, „game-changer“. Kapitola 2 nám dala nástroj, jak ten křik proscreenovat: **detektor Whitehouse/Kelvin**.

Připomeňme si ho. Whitehouse chtěl protlačit signál transatlantickým kabelem *silou* — vyšším napětím — a zabil kabel. Kelvin dostal tutéž zprávu skrz *citlivějším měřením* — jemným galvanometrem. Otázka na každý nový nástroj tedy zní: *přidává řešiteli hrubou sílu, nebo mu zostřuje měření a úsudek?*

Prožeňme tím naši dílnu poctivě. Agent, který za minutu napíše pexeso, je *síla* — obrovská, laciná, svůdná. Sama o sobě je to Whitehousovo napětí: naleje do problému víc kódu, rychleji. Kdyby to bylo všechno, byla by to past. Co z těch nástrojů dělá Kelvina, není generování — je to *diff, plán a commit*. To jsou přístroje na měření: ukazují ti, co přesně stroj udělal, tah po tahu, dřív než tomu uvěříš. Nástroj, který ti jen chrlí kód a neukáže diff, je čistý Whitehouse a máš před ním couvnout. Nástroj, který ti každý svůj tah předloží ke kontrole, ti přidává citlivost — a teprve ta z laciné síly dělá řemeslo.

→ *kap. 2: detektor Whitehouse/Kelvin. Síla bez měření zabíjí (kabel); měření dostane zprávu tam, kam síla nedosáhla. Aplikuj na* každý *příští nástroj, který uvidíš — a uvidíš jich do konce roku ještě deset.*

Takže test není „jak rychle to generuje“. Test je: *dovolí mi to ověřit, co udělalo?* Rychlost generování je Whitehousovo napětí; čitelný diff je Kelvinův galvanometr. Kupuj galvanometry.

Kam to míří

Poskládej to dohromady. Nápad ventiluješ do hotové věci tím, že ho položíš vlastními slovy stroji, který sáhne po nástrojích, ukáže ti diff a nechá tě rozhodnout, jestli je to dost dobré na commit. Syntaxi napíše — to už není tvoje práce od sedmnácté kapitoly. Tvoje práce jsou konce toho pohybu: dobře položit úlohu a poznat, že výstup je špatně.

Dílny jsou dvě. Terminál, když chceš skládat a opakovat; editor, když chceš vidět a klikat. Terminál vyhrává tam, kde jde o kompozici, ze stejného důvodu jako roura nad klikáním — je to článek, ne konec cesty. A na každý nový nástroj, který v téhle rychlé době potkáš, máš detektor: neptej se, jak rychle generuje, ptej se, jestli ti dovolí ověřit, co vygeneroval.

Zbytek dějství tenhle jediný pohyb rozebírá do hloubky. Příští kapitola dá agentovi *ruce a oči* — nástroje na okolní svět (MCP) a pojistky, aby ti neublížil (hooks, sandbox). Pak mu předáš *své řemeslo* (skills), naučíš se *nepustit ho na produkci naslepo* (git v praxi), a nakonec všechno spojíš do jediného oblouku — od nápadu k běžící produkci. Tohle byla mapa dílny. Teď vezmeme nářadí do ruky.

„Jak bych poznal, že se pletu?“

Konkrétní test k této kapitole: až příště sáhneš po novém nástroji, co „píše kód za tebe“, nezkoumej, jak rychle generuje — to je Whitehousovo napětí a svede tě. Zkus místo toho jedinou věc: *požádej ho o změnu a najdi, kde ti ukáže diff.* Když ti tah po tahu předloží, co udělal, k odsouhlasení — přidal ti Kelvinovu citlivost a nástroj obstál; smíš ho pustit dál. Když ti jen vysype hotový výsledek bez možnosti zkontrolovat, co přesně změnil, je to čistá síla bez měření — a kapitola tě varovala, abys couvl. Pletl jsem se jen tehdy, když najdeš nástroj, který generuje bleskově *a přitom* ti každý svůj tah dá k ověření tak srozumitelně, že ho zvládneš přečíst rychleji, než jsi psal prompt — pak jsi našel lepší galvanometr, než jsem znal já, a napiš mi o něm.