XXV

Jak modelu předáš své řemeslo?

Po této kapitole budeš vědět, že „naučit„ model novou dovednost není programování v žádném jazyce, který znáš — je to napsání jednoho textového souboru. A budeš vědět, že o tom, jestli ta dovednost vůbec ožije, nerozhoduje kód uvnitř, ale jediná věta, kterou ji popíšeš.

> *Věnuj zvláštní pozornost poli* `name` *a* `description` *svého skillu. Podle nich se Claude rozhoduje, zda skill v dané úloze spustí.*
>
> — Barry Zhang, Keith Lazuka, Mahesh Murag (Anthropic), *„Pay special attention to the* `name` *and* `description`*... Claude will use these when deciding whether to trigger the skill.“* · Equipping agents for the real world with Agent Skills, *anthropic.com/engineering, 16. 10. 2025*

Soubor, který nikdo nezkompiluje

Chtěl jsem, aby mi model dělal jednu opakovanou věc přesně tak, jak ji dělám já: brát vědeckou studii a překlápět ji do slajdů ve stejném stylu — otázka, důkaz, shrnutí, žádné reklamní titulky. Podesáté jsem do chatu vlepil tentýž odstavec instrukcí a podesáté jsem se přistihl, jak ho ručně upravuji, protože model pokaždé zapomněl, že korelace není kauzalita. Napadlo mě, co asi každého: *musí přece jít tuhle dovednost do modelu jednou provždy zabudovat.* Čekal jsem volání nějakého API. Klíč. Trénování. Něco těžkého.

Ono to bylo tohle. Založil jsem složku a do ní jediný textový soubor:

```
~/.claude/skills/veda-vs-media/SKILL.md
```

Do souboru jsem napsal — obyčejnou češtinou, žádný programovací jazyk — co ta dovednost dělá, kdy se má použít a jak postupovat. Uložil. A při dalším dotazu model tu dovednost sám poznal, sám ji použil a slajdy přišly ve správném stylu, aniž jsem cokoli vysvětloval. Nic se nezkompilovalo. Nic se nenainstalovalo. Řemeslo, které jsem podesáté opisoval do chatu, jsem předal modelu tím, že jsem ho jednou napsal do Markdownu. Tomuhle souboru se říká *skill*, a tahle kapitola

je o jediné otázce, která u něj rozhoduje o všem: proč jednou větou stojí a druhou padá.

Model si tě vybírá sám

Tady je věc, která mi trvala dlouho, než mi doopravdy došla, a všechno ostatní z ní plyne: *ten skill nikdo nezavolá.* Ty ne, a — skoro nikdy — ani ty přes příkaz. Zavolá si ho model sám. A rozhodne se pro to na základě jediné věci, kterou o skillu ví, dokud ho nepoužije: jeho *popisu*.

Představ si model jako řemeslníka, který vejde do dílny plné zásuvek. Na každé zásuvce je štítek — jedna věta, co je uvnitř. Řemeslník do zásuvek nevidí; vidí jen štítky. Když dostane úkol, přečte štítky, a podle nich se rozhodne, kterou zásuvku otevře. Když je štítek matný nebo lže, zásuvka zůstane zavřená, i kdyby v ní ležel přesně ten nástroj, který úloha potřebuje. A obráceně: když je štítek příliš lačný a slibuje víc, než uvnitř je, řemeslník otevře zásuvku i tam, kde neměl, a plete se do věcí, do kterých mu nic není.

Ten štítek je pole `description`. A tady je celý paradox skillů: můžeš dovnitř napsat sebedokonalejší postup, sebechytřejší skripty, sebelepší řemeslo — když je štítek špatný, model zásuvku *nikdy neotevře* a tvoje práce leží ve tmě. Jedna věta rozhoduje o bytí a nebytí celé dovednosti. Ne kód. Věta.

> Můj první vlastní skill se nespouštěl. Uvnitř byl poctivý postup, hodiny práce. Zkoušel jsem prompt za promptem a model si ho pořád nevšímal — sahal vedle, do zásuvky, kterou jsem nechtěl. Půl večera jsem hledal chybu *uvnitř*. Nebyla tam. Byla ve štítku: napsal jsem `description: "nástroj na slajdy"`. Tři slova, do kterých se nevešlo *kdy* ho použít. Model neměl podle čeho poznat, že „rozeber mi tu studii„ je přesně ono. Přepsal jsem tu jednu větu — a skill ožil. Půl večera za jednu větu, kterou jsem měl napsat pořádně na začátku.

Zapamatuj si to jako první zákon skillů, protože se k němu ještě několikrát vrátíme: *skill se nespouští tím, co umí, ale tím, jak je popsaný.*

skill — složka s jedním povinným souborem `SKILL.md` *(instrukce v Markdownu*
1. *krátká hlavička) a volitelnými scripty, referencemi a šablonami. Otevřený standard* Agent Skills *(agentskills.io), který vedle Claude Code přijaly i další nástroje. Ohlášen Anthropic 16. 10. 2025.*

Anatomie jednoho souboru

TEENAGER · MECHANISMUS

Otevři `SKILL.md` a uvidíš dvě části oddělené třemi pomlčkami. Nahoře je *hlavička* (frontmatter) — pár řádků, které model čte jako první. Pod ní je *tělo* — obyčejný Markdown, tvoje instrukce.

```
---
name: veda-vs-media
description: Rozebere vědeckou studii a vytvoří
slajdy ve stylu
  otázka → důkaz → shrnutí. Použij, když
uživatel chce rozebrat
  studii, odhalit zkreslení v médiích, nebo
porovnat titulek
  s tím, co studie doopravdy říká.
---

# Věda vs. média

## Kdy použít
- „rozeber tu studii", „udělej slajdy o X"
- uživatel chce ukázat rozdíl mezi titulkem a
realitou

## Postup
1. Rozliš typ studie (pozorovací ×
randomizovaná).
2. Odděl absolutní riziko od relativního.
3. Sestav slajdy: otázka → důkaz → shrnutí.
```

Dvě pole v hlavičce jsou celá duše skillu. `name` je jméno — pod ním ho taky ručně zavoláš jako `/veda-vs-media`. `description` je ten štítek na zásuvce: *co* skill dělá a hlavně *kdy* ho použít. Zbytek hlavičky je nepovinný. Tělo je řemeslo — přečte se, teprve když model podle štítku rozhodne, že skill použije.

Povinný je jen `SKILL.md`*; ostatní soubory jsou volitelné.* `name` *slouží i jako jméno slash-příkazu (*`/název`*). Doporučení Anthropic: tělo* `SKILL.md` *drž pod 500 řádky — cokoli delšího odsuň do* `references/` *a načti až v případě potřeby.*

Vedle souboru můžou být tři složky a stojí za to je rozlišit, protože každá řeší jiný problém. `scripts/` je na věci, které se nemají *vymýšlet* pokaždé znovu: hotový skript, který model spustí, je rychlejší a spolehlivější než kód, který by model naráz plodil z hlavy. `references/` je na velké, občas potřebné znalosti — dokumentace API, tabulky, schémata; vejdou se sem klidně tisíce řádků, protože se načtou, jen když jsou zrovna třeba. `assets/` jsou hotové soubory, které skill přímo použije ve výstupu — šablona dokumentu, fonty, obrázky. Řemeslo do těla, velké znalosti do referencí, ruční práce do skriptů, materiál do assets.

TEENAGER · MECHANISMUS

Kde skill hledat a kam ho položit. Model prohledává tři místa a čím konkrétnější, tím užší dosah:

```
~/.claude/skills/<název>/     # osobní — napříč
všemi tvými projekty
.claude/skills/<název>/       # projektový —
verzovaný v gitu, sdílený s týmem
(uvnitř pluginu)              # pluginový —
přijde hotový z marketplace
```

Projektový skill je ten zajímavý: leží v repozitáři, jde do commitu (kap. 18) a tím se *řemeslo stane součástí projektu*. Nový člen týmu — nebo agent, který v repozitáři pracuje — dostane tvoje postupy zadarmo, protože jsou verzované vedle kódu. Skill není osobní makro; je to sdílený artefakt.

Tři patra: proč se toho nenačte moc

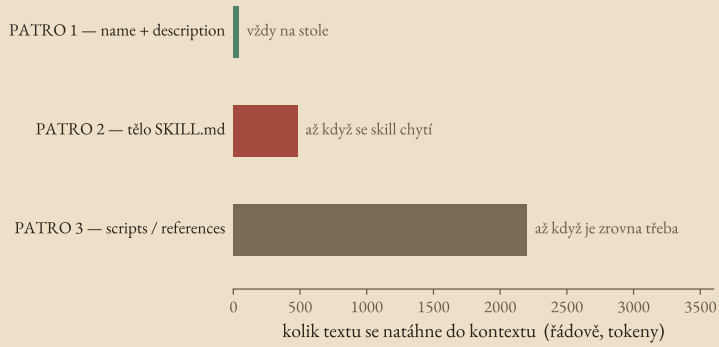
TEENAGER · MECHANISMUS

Kdyby model musel mít *všechny* skilly celé v hlavě naráz, utopil by se — pracovní stůl (kontextové okno z kap. 15) má jen tolik místa. Skilly to řeší chytře: načítají se *na patra*, a skoro pořád je na stole jen to úplně nejlehčí.

Patro 1 — vždycky. Z každého nainstalovaného skillu leží na stole jen `name` a `description`. Pár desítek tokenů na skill. Tohle model čte, aby se rozhodl.

Patro 2 — když se skill chytí. Teprve když model podle popisu usoudí, že skill patří k úloze, natáhne celé tělo `SKILL.md`.

Patro 3 — když je zrovna třeba. Skripty a reference se načtou, jen když na ně v postupu dojde. Do té doby nestojí skoro nic.



Tomuhle se říká progresivní (líné) načítání*: nenatahuj nic, dokud to nepotřebuješ. Proto můžeš mít nainstalovaných sto skillů a stůl přitom zůstane skoro prázdný — model nese jen sto štítků, ne sto plných manuálů. Model neví nic z toho, co mu zrovna neleží na stole; skilly hospodaří přesně s touhle vlastností.*

Podívej se na ten obrázek, protože je v něm celé kouzlo. Patro 1 je ten tenký verdigrisový proužek — pár desítek tokenů, které leží na stole *pořád*, za každý skill. Patro 2 se natáhne, teprve když se skill chytí. Patro 3 čeká na disku, dokud na něj nedojde. A teď dosaď první zákon: jediné, co model o skillu ví ve chvíli rozhodování, je ten tenký proužek nahoře — `name` a `description`. Proto na něm všechno stojí. Rozhoduje se z patra 1; řemeslo z patra 2 a 3 se uplatní jen tehdy, když ho patro 1 pustí ke slovu.

A tady je ta dobrá zpráva, kvůli které se to vůbec vyplatí: skill napíšeš *jednou* a od té chvíle ho model nosí s sebou, aniž tě to na stole cokoli stojí. Nemusíš podesáté lepit tentýž odstavec do chatu. Řemeslo, které jsi léta nosil v hlavě, teď leží ve složce — a model si ho vezme přesně tehdy, když je ho potřeba. To je úžasné; jen si to musíš pojistit tou jednou větou.

Skill, prompt, nebo MCP?

Skill je jeden ze tří způsobů, jak modelu předat něco navíc, a lidé si je pletou. Rozdíl je snadný, jakmile ho jednou uvidíš.

Prompt je věta, kterou napíšeš teď a platí do konce hovoru. Skvělý na jednorázovou věc. Ale zítra je pryč a příště ho musíš napsat znovu — přesně ta opakovaná ruční práce, kterou jsi podesáté lepil do chatu. Skill je prompt, který jsi jednou uložil a pojmenoval; model si ho bere sám, kdykoli se hodí, a přežije konec hovoru i restart.

MCP (kap. 24) dává modelu *ruce* — připojení k živému systému: sáhne do databáze, zavolá službu, přečte tvoje soubory. Skill dává modelu *znalost postupu*: jak něco udělat, v jakém pořadí, na co si dát pozor. MCP je zásuvka s nářadím napojeným na svět; skill je návod, jak s nářadím pracovat jako mistr. Často jdou ruku v ruce: MCP připojí databázi, skill popíše, jak z ní poctivě tahat čísla, aby ses neoklamal.

Hrubé pravidlo: opakuješ tentýž postup → skill. Model potřebuje sáhnout do živého systému → MCP. Jednorázová věc pro tenhle jeden hovor → prompt. MCP stojí místo na stole pořád (nese popisy nástrojů); skill v patře 1 skoro nic — proto se sto skillů unese líp než deset upovídaných MCP serverů.

Kdo skill spustí — a jak si to pojistíš

TEENAGER · MECHANISMUS

Skill se vyvolá dvěma cestami. *Automaticky*: model čte `description`, a když sedí na úlohu, spustí ho sám. To je běžný případ a to je ono „model si tě vybírá sám„. *Ručně*: každý skill je zároveň slash-příkaz, takže napíšeš `/veda-vs-media` a spustíš ho, ať model „cítí“ cokoli.

Obě cesty jde v hlavičce přiškrtit. U skillů *s následky* — které nasazují, mažou, utrácejí — nechceš, aby se model rozhodl sám:

```
---
name: nasad-do-produkce
description: Nasadí aplikaci do produkce.
disable-model-invocation: true    # jen ruční /
nasad-do-produkce
allowed-tools: Bash(git:*), Bash(docker:*)
---
```

`disable-model-invocation: true` znamená: *model tě sám nespustí, čekej na můj příkaz.* Nechceš, aby model nasadil do produkce jen proto, že prošly testy — to je tvoje rozhodnutí, ne jeho (spojnice s kap. 24: sandbox versus souhlas). `allowed-tools` naopak předem povolí nástroje, které skill smí použít bez ptaní.

→ kap. 26: pravidlo „každý tah proti VCS„ platí i pro skilly. Skill, který něco mění, patří za `disable-model-invocation` *a spouští se ručně, s commitem před ním. Odvaha je levná, jen když existuje undo.*

Věž: spuštění jako vyhledávání

EINSTEIN · MATEMATIKA — přeskočitelné

Selekce skillu je retrieval. Formálně. Model má množinu skillů S; ke každému $s \in S$ zná jen jeho popis d_s (patro 1). Přijde úloha — prompt q. Rozhodnutí „spustit skill s„ je odhad podmíněné pravděpodobnosti

$$P(\text{spuštění} = s \mid q, d_s),$$

tedy: *jak dobře popis d_s sedí na úlohu q.* Skill neožívá podle toho, co je v jeho těle — tělo model ve chvíli rozhodování nevidí. Ožívá podle *shody popisu s dotazem*. To je přesně úloha vyhledávání (retrieval): d_s hraje roli indexu, q roli dotazu, a model vrací ten záznam, jehož index dotazu nejlépe odpovídá. Špatný `description` je špatně zaindexovaný záznam — existuje, ale nikdo ho nenajde.

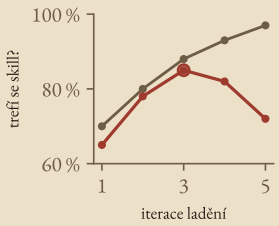
Odtud plynou obě selhání přímo. Přehnaně úzký *popis* → P *je malé i tam, kde skill patří* → *skill „nejde chytit„ (moje jizva).* Přehnaně lačný *popis* → P *je velké i tam, kam skill nepatří* → *falešné spuštění, plete se do cizího. Ladit description je posouvání těhle pravděpodobností nahoru tam, kam patří, a dolů tam, kam ne — přesně kompromis přesnost × úplnost z kap. 12.*

EINSTEIN · MATEMATIKA — přeskočitelné

Description se dá přeučit — a jak to poznáš. Popis se dá ladit, a dá se ladit *měřením*, ne citem. Sestavíš dvě sady promptů: takové, na které *má* skill naskočit, a takové, na které *nemá*. To jsou tvá data. Pak popis přepisuješ a měříš, kolik z nich trefí. Přesně jako u modelu z kap. 11 hrozí *přeučení*: vyladíš `description` tak, že sedí na tvoje trénovací prompty skvěle — a na nových, které jsi neviděl, spadne.

$$\text{úspěšnost}_{\text{train}} \uparrow, \quad \text{úspěšnost}_{\text{test}} \downarrow .$$

Proto sadu rozděl na trénovací a *testovací* (kap. 11: testovací data jsou svatá — kdo na nich ladí, klame sám sebe). Ladíš na trénovacích, rozhoduješ podle testovacích. Skutečná ukázka z ladicí smyčky, kde se `description` optimalizuje automaticky:



Šedá (trénink) roste pořád — čím víc description mučíš, tím líp sedí na prompty, které jsi viděl. Červená (test) nejdřív roste s ní, pak spadne: v iteraci 5 sedí popis na trénink na 97 %, ale na held-out promptech jen na 72 %. Vítěz je kroužek v iteraci 3 — ne poslední, nejlepší na testu. *Papoušek versus student (kap. 11), přenesený na jednu větu štítku.*

Poskládej to dohromady. Skill je textový soubor, který nikdo nekompiluje. Model o něm ve chvíli rozhodování ví jen jednu věc — popis — a podle něj sáhne, nebo nesáhne. Proto řemeslo uvnitř nerozhoduje o spuštění; rozhoduje o něm ta věta na štítku. A protože je to vyhledávání, dá se měřit a přeučit jako každý jiný model — takže se ladí na datech a soudí na testu, ne citem. Předat modelu řemeslo je snadné. Napsat větu, díky které si ho v pravou chvíli vezme — a jen tehdy — je to celé řemeslo téhle kapitoly.

DEMO

`lesk-a-bida-vibecodingu.gaussalgo.com/d/skill-trigger`

Napiš `description` a hra pustí dvacet promptů — deset, co mají skill chytit, deset, co nemají. Uvidíš svou přesnost a úplnost naživo a to místo, kde ladění přeteče v přeučení.

„Jak bych poznal, že se pletu?“

Konkrétní test k této kapitole — *trigger test*. Nevěř tomu, že tvůj skill funguje, dokud jsi to nezměřil takhle: napiš pět promptů, na které *má* naskočit, a pět, na které *nesmí* (např. „udělej mi obyčejnou prezentaci o dovolené“ — tam věda-vs-média nemá co dělat). Pusť je. Skill je dobrý jen tehdy, když chytil všech pět správných a ani jeden z pěti špatných — a když ne, chyba *skoro nikdy* není v těle, ale v popisu, tak ho přepiš a změř znovu. Kapitola se plete jen tehdy, když se ukáže, že model spouští skill přesně podle úlohy, i když jsi `description` napsal ledabyle — pak spouštěč nebydlí v popisu a první zákon téhle kapitoly padá. (Zatím vždycky bydlel tam.)