XXVI

Jak nepustíš agenta na produkci naslepo?

Teorie z kapitoly osmnácté teď v ruce. Naučíš se pouštět agenta z řetězu odvážně, ne bezbranně: commit jako pojistka, worktree jako klec pro paralelní agenty, `/diff` jako lupa a bisect jako detektiv na bug, který ti tam nechal stroj.

> *Moje zlaté pravidlo pro produkční programování s AI zní: nekomitnu do repozitáře žádný kód, u kterého bych nedokázal někomu přesně vysvětlit, co dělá.*
>
> — Simon Willison, *„Not all AI-assisted programming is vibe coding (but vibe coding rocks)“* · simonwillison.net, *19. 3. 2025*

Začnu scénou, kterou možná znáš z vlastní kůže. Sedneš k projektu, který zhruba funguje, ale je v něm pár míst, co ti dlouho vadí. Napíšeš agentovi: „přepiš prosím celou práci s uživateli, ať je to čistší.“ Odejdeš si pro kávu. Vrátíš se za dvacet minut a agent zářivě hlásí, že hotovo — přepsal dvě stě řádků, „výrazně to zjednodušil“, refaktoroval tři soubory a smazal jeden, který podle něj „už nebyl potřeba“. Otevřeš aplikaci. Přihlášení nefunguje. Půlka toho, co udělal, je vážně lepší než předtím. Druhá půlka tiše rozbila věci, které fungovaly roky.

A teď dvě verze konce téhle scény. V první jsi před spuštěním agenta neudělal nic — jen jsi mu předal řízení. Teď sedíš nad projektem, který nepoznáváš, protože ho přepsal *přes* tvůj poslední stav, a snažíš se z hlavy rekonstruovat, co tam bylo dřív. Večer je pryč. Ve druhé verzi jsi třicet vteřin předtím napsal jediný příkaz:

```
git commit -am "před refaktorem uživatelů"
```

A rozdíl je propastný. Ať agent nadělal cokoli, jsi *jeden příkaz* od stavu před ním. Vytáhneš si v klidu jen ty dobré kusy jeho práce, zbytek zahodíš, a za pět minut pokračuješ. Celá tahle kapitola je o té třicetivteřinové propasti — a o dalších pojistkách stejného druhu.

→ *kap. 18: tam jsme git poznali jako* stroj času *— commit je uložená pozice ve hře, větev paralelní vesmír, merge soud. Tady tytéž pojmy přestávají být teorií a stávají se reflexem, kterým chráníš práci před strojem, co píše rychleji, než stačíš číst.*

Batole: každý experiment agenta musí být vratný

Vezmi si tuhle jednu větu a odejdi s ní, i kdybys nečetl nic dalšího: *dokud existuje undo, smíš zkoušet cokoli.* Práce s agentem je totiž hazard jen tehdy, když je nevratná. Ve chvíli, kdy víš, že se vždycky vrátíš do známého stavu, se z hazardu stane experiment — a experimentovat je levné, zábavné a poučné.

Odvaha není povahový rys, který někdo má a jiný ne. Odvaha je *levná, když existuje záchranná síť.* Provazochodec nad propastí je hrdina; provazochodec metr nad matrací jen zkouší, kolik unese, a když spadne, vstane a zkusí to líp. Git je ta matrace. Než pustíš agenta na cokoli většího, uděláš commit — a tím jsi přesně metr nad matrací. Můžeš agentovi dovolit divoké nápady, protože nejhorší, co se stane, je, že se vrátíš o krok zpátky.

Tohle je celé batolecí patro téhle kapitoly, a je to nejdůležitější věc z ní: *nikdy nepouštěj agenta na stav, do kterého se neumíš vrátit.* Všechno ostatní — worktrees, bisect, review — jsou jen chytřejší způsoby, jak tuhle jednu zásadu dotáhnout do každodenní praxe.

A dobrá zpráva je, že ta síť tě nic nestojí. Commit je dvě vteřiny a jeden řádek. Za tu cenu kupuješ svobodu říct agentovi „zkus to úplně jinak“ bez bušení srdce. Nástroje jsou dnes opravdu úžasné — pustit agenta přepsat půl projektu je zážitek. Ale úžasné jsou teprve tehdy, když pod ně napneš síť, do které smíš spadnout.

Teenager: čtyři pojistky, které používáš každý den

TEENAGER · MECHANISMUS

Batolecí pravidlo „všechno vratné“ se v praxi rozpadá na hrstku návyků. Žádný není složitý; síla je v tom, že je děláš *pokaždé*, ne jen když si vzpomeneš.

```
# 1) COMMIT PŘED KAŽDÝM "PŘEPIŠ TO CELÉ"
#    Reflex, ne rozhodnutí — jako pás dřív, než
nastartuješ.
git commit -am "checkpoint před refaktorem
plateb"

# 2) .gitignore — co agentovi ani gitu nikdy
nedáš do ruky
echo ".env"          >> .gitignore   # klíče a
hesla
echo "node_modules/" >> .gitignore   # stažené
závislosti
echo ".claude/settings.local.json" >> .gitignore

# 3) /diff PŘED každým souhlasem — přečti, co
agent napsal
#    (v Claude Code, Antigravity CLI i jinde:
ukáže tah po tahu)

# 4) VĚTEV NA EXPERIMENT — hlavní tok zůstane
čistý
git switch -c pokus-nova-architektura
#   ... pusť agenta řádit ...
git switch main         # nepovedlo se? Vrať se,
větev zahoď
```

Pravidlo 1 je páteř. Ostatní tři jen řeší situace, kdy commit sám nestačí: co agentovi vůbec ukázat (2), jak zkontrolovat, co udělal (3), a jak nechat několik pokusů běžet vedle sebe, aniž se popletou (4).

`git commit -am` — `-a` *přidá do commitu všechny změněné sledované soubory,* `-m` *připne vzkaz. Jeden řádek, žádné* `git add`*. Pro rychlý checkpoint před experimentem ideální; pro čistou atomickou historii commituj raději po částech (viz kap. 18).*

.gitignore chrání dvakrát: jednak aby se tajemství nedostala do historie (odkud je nesmažeš), jednak aby agent nemarnil kontext čtením `node_modules`*. Co jednou commitneš, zůstává ve stroji času navždy — klíč zveřejněný v historii je klíč zveřejněný.*

Worktrees: klec pro každého paralelního agenta

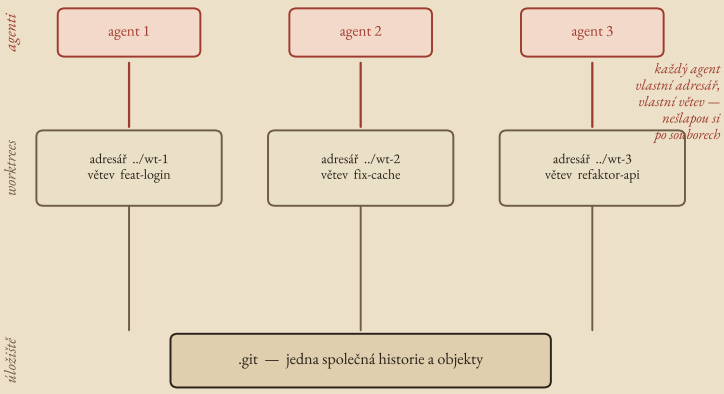
TEENAGER · MECHANISMUS

Tady je trik, který dělá z jednoho člověka malý tým. Agenti umí běžet paralelně — pustíš jednoho na přihlašování, druhého na cache, třetího na refaktor API. Jenže když jim dáš *jeden adresář*, šlapou si po souborech: agent 2 uloží rozdělanou práci, agent 1 ji přepíše, oba se zaseknou na zamčeném souboru a máš chaos. Řešení je `git worktree`:

```
# z hlavního repozitáře vyrob tři pracovní
adresáře, každý na své větvi
git worktree add ../wt-login   -b feat-login
git worktree add ../wt-cache   -b fix-cache
git worktree add ../wt-api     -b refaktor-api

git worktree list        # co kde běží
#   ... v každém adresáři pustíš vlastního
agenta ...
git worktree remove ../wt-login   # hotovo,
uklidíš
```

Každý adresář má vlastní soubory a vlastní větev, ale všechny sdílejí *jednu* historii — společné úložiště `.git`. Agenti si tak nemůžou navzájem přepsat rozdělanou práci (izolace na disku), a přitom výsledky sliješ přes obyčejný merge, protože historie je společná. Je to paralelní vesmír z kapitoly osmnácté, jen zhmotnělý do tří adresářů, do kterých se dá zároveň sáhnout.



Hrdinský graf kapitoly. Dole jediné úložiště `.git` — jedna společná historie a všechny objekty. Nad ním tři worktrees: samostatné adresáře, každý přepnutý na vlastní větev. Nahoře agent v každém z nich. Šipky vedou ke společnému kořeni, ale adresáře se nepřekrývají — proto si tři agenti nešlapou po souborech, i když jedou naráz.

Jedno varování, ať tě to nezaskočí: worktree izoluje *soubory*, ne *běh*. Když každý agent spustí vlastní testovací server na tomtéž portu, o databázi nebo o port se poperou dál — izolace disku není izolace prostředí. Na malé věci to nevadí; jak porostou, přidáš každému agentovi vlastní

port a vlastní databázi. Ale to je detail; hlavní zisk — že si nepřepíšou kód — máš zadarmo.

Claude Code umí worktree provozovat i sám: subagentu se v hlavičce nastaví `isolation: worktree` *a orchestrátor mu vyrobí čerstvý adresář, který po dokončení uklidí (dle* Claude Code Docs, *k 7/2026). Princip znej i tak — ať víš, co se pod tebou děje.*

/diff a review: kde se do řetězu vrací tvůj úsudek

Teď to nejtěžší, protože to není příkaz, ale disciplína. Agent ti vygeneruje kód plynule — a plynulost není správnost. Model se nestydí a negratuluje si k nesmyslu; napíše sebejistě funkci, která *vypadá* hotově, a přitom tiše počítá slevu z už zlevněné ceny. Jediné místo, kde se do řetězu vrací *tvůj* úsudek, je chvíle, kdy si to přečteš dřív, než tomu uvěříš.

pull request (PR) — návrh na začlenění větve, místo, kde proběhne review; na sdíleném projektu ho čte kolega, na svém ho čteš ty sám. Pro AI výstupy je review dvojnásob povinné: u lidského kolegy předpokládáš stud a intuici, u modelu ani jedno (kap. 16 — model plynule doplní i to, co je špatně).

Nástroje ti to usnadňují: `/diff` v Claude Code i v Antigravity CLI ukáže, co agent změnil proti gitu — soubor po souboru, tah po tahu. Přečti to. Ne proto, že ti někdo přikazuje buzeraci navíc, ale protože je to tvoje jediná obrana proti tomu, aby ti do repozitáře vtekl kód, kterému nerozumíš. A přesně tady platí Willisonovo zlaté pravidlo z čela kapitoly beze zbytku: *nekomitni nic, co bys neuměl vysvětlit.* Když to neumíš vysvětlit, ještě to nechápeš — a co nechápeš, nemůžeš ověřit.

Proč u AI dvojnásob? Protože u lidského kolegy review chytá *omyly* — chvíle únavy, překlepy, nedomyšlené kraje. U člověka je základ v pořádku, review je jemný filtr. U agenta chybí i ten základ: nemá představu, co je v projektu posvátné, které funkce nesmí sáhnout, co znamená „hotovo“ pro *tebe*. Vygeneruje věrohodně vypadající celek, ve kterém je správné devět desetin a katastrofa v desetině — a ta desetina se nepozná plynulostí, jen čtením. Čím je model lepší, tím je to zrádnější: špatná odpověď od hloupého modelu je vidět na první pohled, špatná odpověď od chytrého modelu vypadá jako dobrá.

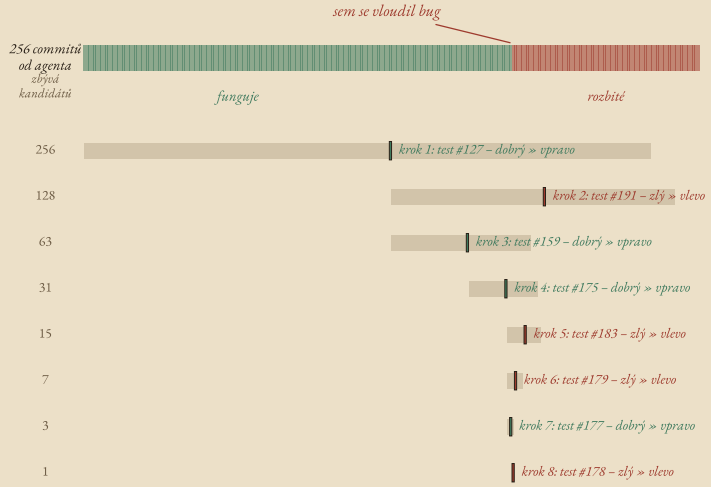
Když už to rozbil: bisect jako detektiv na bug od agenta

Řekněme, že jsi commitoval poctivě, pustil agenta na několik sezení a on mezitím udělal spoustu drobných commitů — agenti commitují často a rád. Dnes zjistíš, že něco nefunguje, a víš dvě věci: *teď* je tam bug a *před dvěma sty commity* tam nebyl. Který z nich ho zavlekl? Projít je jeden po druhém od konce by znamenalo klidně dvě stě testů. Git to udělá půlením — `git bisect`:

```
git bisect start
git bisect bad                # tenhle stav je
rozbitý
git bisect good v1.0          # tenhle byl v pořádku
#   git tě přepne doprostřed intervalu; otestuješ a
řekneš good/bad
git bisect good               # ...nebo bad, podle
výsledku
#   git zase půlí; opakuješ, dokud nezbude jediný
```

```
commit
git bisect reset              # hotovo, vrať se, kde
jsi byl
```

Každá odpověď zahodí *polovinu* zbylých kandidátů. Z dvou set padesáti šesti commitů najdeš viníka na *osm* testů, protože dvě na osmou je dvě stě padesát šest. Je to táž myšlenka jako hádání čísla „výš — níž“: kdo půlí, ten neprohledává, ten *vylučuje*.



osm testů místo dvě stě padesáti šesti — a s testem běží bisect sám

Nahoře všech 256 commitů od agenta v čase: zelené fungují, sanguine jsou rozbité, a někde mezi nimi je hrana — ten jeden commit, co bug zavlekl. Každý řádek dole je jeden krok bisectu: git testuje prostředek zbylého intervalu, tvá odpověď utne půlku, pásmo kandidátů se scvrkává 256 → 128 → 63 → 31 → 15 → 7 → 3 → 1. Osm testů místo dvou set padesáti šesti.

A tady se dvě pojistky spojí v jednu. Pokud na ten bug máš *test* — kousek kódu, který sám řekne „funguje / nefunguje“ —, bisect nemusíš klikat ručně:

```
git bisect start HEAD v1.0
git bisect run pytest tests/test_prihlaseni.py
#   git projede historii SÁM a vyplivne vinný commit
```

Testy plus bisect se rovná automatický detektiv, který ti najde, kde přesně se to pokazilo, bez tvého zásahu. A to je krásné vyústění celé věci: čím poctivěji agent commitoval a čím lepší testy máš (o těch je příští dějství), tím rychleji najdeš jeho chybu. Nepořádek se mstí, pořádek se vyplácí — a u stroje, který dělá dvě stě commitů za odpoledne, se vyplácí dvojnásob.

→ kap. 28: `git bisect run <test>` je proč se vyplatí psát testy dřív, než pustíš agenta. Bisect (kde) + test (jestli) = detektiv, který najde vinný commit bez tvého klikání — i mezi stovkami commitů, které nadělal stroj.

DEMO

`lesk-a-bida-vibecodingu.gaussalgo.com/d/bisect`

Bisect hra: schoval jsem bug do jednoho ze 128 commitů, jako by ho tam nechal agent. Najdi ho — a sleduj, jak ti každá odpověď „good/bad“ půlí zbytek. Zvládneš to na sedm kroků?

Einstein: proč worktree izoluje a proč je review u AI jiná disciplína

EINSTEIN · MATEMATIKA — přeskočitelné

Proč worktree stačí na izolaci souborů — a kde končí. V kapitole osmnácté jsme viděli git jako trojici stavů: *working tree* (soubory na disku), *index* (co půjde do příštího commitu) a *HEAD* (špička větve, na které stojíš). Klíč k worktrees je, že tahle trojice je *per-worktree*, kdežto úložiště objektů (adresář `.git`) je *sdílené*. Formálně: každý linked worktree má vlastní (W, I, HEAD), ale všechny sdílejí jeden objektový graf — týž Merkle DAG obsahově adresovaných objektů z kapitoly osmnácté.

Důsledek je přesně ta izolace, kterou chceš pro paralelní agenty. Dva agenti ve dvou worktrees nemůžou způsobit konflikt na disku, protože každý zapisuje do svého W a svého I; sdílejí jen *neměnné* objekty, které se adresují obsahem, a neměnný objekt nejde zkazit souběhem. Kde izolace končí, je všechno *mimo* tuhle trojici: porty, běžící procesy, databáze, dočasné soubory vně repozitáře. Ty git nespravuje, takže je ani neizoluje. Proto worktree = izolace stavu verzí, nikoli izolace běhového prostředí — a proto pro plnou izolaci paralelních agentů přidáváš kontejner (kap. 27), který ohradí i to zbývající.

linked worktree — přídavný pracovní adresář připojený k témuž repozitáři. Sdílí objektové úložiště a odkazy, ale má vlastní HEAD *a index. Proto tři agenti ve třech worktrees pracují nezávisle, a přitom jejich výsledky sliješ jedním mergem — historie je od začátku jedna.*

EINSTEIN · MATEMATIKA — přeskočitelné

Proč je review u AI kvalitativně jiná úloha než u člověka. Zkusme to vyjádřit jako poměr. U lidského kolegy je základ v pořádku a review chytá vzácné omyly; podíl řádků, které musíš skutečně zpochybnit, je malý — řekněme $p_{\text{člověk}}$. U agenta ale nesdílíš kontext o tom, co je v projektu posvátné, a model generuje plynule i mimo svůj skutečný dosah; podíl řádků, které je nutné zpochybnit, p_{agent}, je vyšší a hlavně *nekoreluje* s tím, jak sebejistě text zní.

A teď zrádná část. Pravděpodobnost, že *uděláš* chybu při review — že přehlédneš vadný řádek —, roste s objemem a plynulostí čteného. Označme ji q. Očekávaný počet propuštěných bugů na jedno review je pak zhruba

$$E[\text{propuštěné bugy}] = N \cdot p \cdot q,$$

kde N je počet řádků. U agenta rostou *oba* rizikové činitele naráz: agent chrlí velké N (přepíše dvě stě řádků, kde bys ty napsal dvacet) a jeho výstup má vyšší p. Součin $N \cdot p$ tak neroste lineárně s pohodlím, které agent slibuje — roste rychleji. Willisonovo pravidlo „nekomitni, co neumíš vysvětlit“ je operační způsob, jak srazit q k nule: nutí tě číst tak pomalu, abys každý řádek opravdu pochopil. Není to morální apel, je to snížení jednoho činitele v součinu, který jinak exploduje právě proto, že je agent rychlý.

Rovnice je hrubý odhad řádové úvahy, ne přesný model — bugy nejsou nezávislé a p, q nejsou konstanty. Ale nese správnou pointu: rychlost agenta zvětšuje N, a jestli s ní nesrazíš q (pomalým, chápajícím čtením), počet propuštěných bugů roste, ne klesá.

Co si odnést

Poskládej to dohromady. Pouštět agenta na kód není hazard, pokud je každý jeho tah vratný — a udělat ho vratným stojí třicet vteřin a jeden commit. *Commit* je záchranná síť, pod kterou smíš spadnout. *Worktree* je klec, ve které běží tři agenti naráz, aniž si přepíšou práci. `/diff` *a review* jsou lupa, kterou vrátíš do řetězu svůj úsudek — u AI výstupu dvojnásob, protože model nemá stud a plynulost není správnost. A *bisect* je detektiv, který v historii nadělané strojem najde vinný commit na hrstku testů, a s testem běží sám.

Nic z toho není nový nástroj — je to git z kapitoly osmnácté, obrácený proti stroji, který píše rychleji, než stačíš číst. Ve světě, kde ti kód generuje agent, není verzování administrativa. Je to *první* věc, kterou si pod tu spolupráci postavíš, dřív než pustíš první prompt. Bez commitu není co vrátit, bez větve není kam experiment schovat, bez `/diff` není co zkontrolovat a bez testu není co dát bisectu. Odvaha pouštět agenty divoce je celá postavená na téhle tiché infrastruktuře pod ní.

→ kap. 27: capstone spojí všechno — spec, build, testy, verzování, kontejner, nasazení. Git je první článek toho řetězu: co není zverzované, nejde otestovat, nasadit ani vrátit, když se produkce zadrhne.

„Jak bych poznal, že se pletu?“

Konkrétní test k této kapitole máš v ruce pokaždé, když otevřeš terminál s asistentem. Než pustíš agenta přepsat něco velkého, zeptej se: *udělal jsem teď commit?* A hned tvrdší druhou otázku: *kdyby to agent zkazil, kolik práce ztratím — minutu, nebo odpoledne?* Když je odpověď „minutu“, pustil jsi ho z řetězu odvážně a smíš zkoušet divoké věci, protože každá je vratná. Když je odpověď „odpoledne“, nepustil jsi ho odvážně — pustil jsi ho *bezbranně*, a kapitola tě právě varovala. A druhý test, na review: vezmi poslední commit, který ti napsal agent, a zkus ho *nahlas* vysvětlit, řádek po řádku, jako bys ho obhajoval před kolegou. Když to nesvedeš, nekomitl jsi kód — komitl jsi něco, čemu věříš, aniž bys to ověřil, a to je přesně ta hranice mezi tím, kdo výsledek umí ověřit, a tím, kdo mu musí věřit. Kapitola se plete jedině tehdy, když ti agent rozbije produkci ze stavu, který jsi commitnul, přečetl a uměl vysvětlit — a vrátit se *přesto* nešlo. V tom případě chci vidět tvůj repozitář, protože to by znamenalo, že stroj času přestal být strojem času.