Member-only story

# Git Worktrees + Claude Code: Parallel AI Development Guide

How I Built a Parallel AI Development Pattern Nobody Talks About

15 min read · 2 days ago

Reza Rezvani Following

Listen Share More

Git Worktrees and Claude Code — The Parallel AI Development Pattern

## Git Worktrees & Claude Code: The Parallel Development Pattern Nobody Talks About

It’s Tuesday morning. You’re deep in an authentication refactor when Slack lights up: production bug, users locked out. You save your work, switch branches, and spend twenty minutes explaining your entire codebase to a fresh Claude session. Fix the five-minute bug. Switch back. Re-explain the auth architecture. Again.

You just spent thirty-five minutes on a five-minute fix.

This happens multiple times daily. If you spend forty percent of your day managing context switches, you’re operating at sixty percent capacity — not because you’re slow, but because the workflow architecture fundamentally breaks parallel work.

After watching our team lose weeks to this pattern, I discovered the bottleneck isn't Claude Cod traditional git branch eliminate that constraint entirely.

This isn't about typing faster or generating more code. It's about removing structural inefficiencies that make simultaneous progress impossible. Here's *(below)* the complete implementation guide, including the problems nobody mentions until you hit them in production.

. . .

## Why Git Worktrees Transform AI-Assisted Development

**Git worktrees allow multiple working trees attached to a single repository.** Unlike cloning — which duplicates the entire `.git` database—worktrees share one repository while maintaining separate working directories for different branches.

**The technical mechanism:** Each worktree contains a `.git` file (not directory) pointing to the main repository's `.git/worktrees/<name>` subdirectory. This architecture enables:

- Shared object database and refs
- Independent working trees per branch
- Isolated file state without duplication
- Concurrent branch development without conflicts

**Why this matters for Claude:** Each worktree provides an isolated execution environment. When you run `claude` in worktree A on the `feature/auth` branch, and simultaneously run `claude` in worktree B on the `hotfix/session` branch, they operate in completely separate filesystem contexts. Neither instance can accidentally modify the other's files. Neither loses context when the other is active.

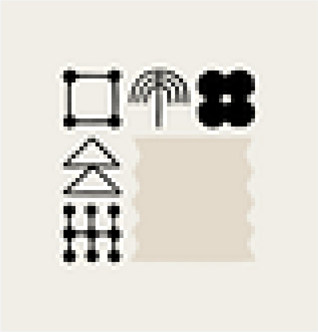
This maps directly to how development actually works: multiple concerns, simultaneous progress, independent contexts. Traditional branch switching breaks this natural pattern. Worktrees restore it.

According to Anthropic’s engineering documentation, worktrees have become their standard o branch manageme

**Claude Code Best Practices**

A blog post covering tips and tricks that have proven effective for using Claude Code across various codebases…

www.anthropic.com



. . .

## Setting Up Your Parallel Workspace

Start from your main project repository. Complete setup takes approximately fifteen minutes.

### Creating Your First Worktrees

```
# Verify clean state
git status

# Create feature worktree
git worktree add ../myproject-auth-refactor -b feature/auth-v2
# Create maintenance worktree
git worktree add ../myproject-bug-session -b hotfix/session-timeout
# Verify structure
git worktree list
```

**Expected output:**

```
/Users/dev/myproject                 abc1234 [main]
/Users/dev/myproject-auth-refactor   def5678 [feature/auth-v2]
/Users/dev/myproject-bug-session     ghi9012 [hotfix/session-timeout]
```

You now have three physical directories, each on its own branch, sharing one Git history.

### Environment Configuration

Each work[illegible]ches:

**Option 1: Copy (for static configs):**

```
cp .env ../myproject-auth-refactor/
cp .env ../myproject-bug-session/
```

**Option 2: Symlink (recommended for dynamic configs):**

```
ln -s $(pwd)/.env ../myproject-auth-refactor/.env
ln -s $(pwd)/.env ../myproject-bug-session/.env
```

Symlinks maintain a single source of truth. Configuration changes propagate automatically.

### Installing Dependencies

**Each worktree needs its own dependency installation:**

```
# JavaScript/TypeScript
cd ../myproject-auth-refactor
npm install
```

```
# Python
cd ../myproject-auth-refactor
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

**Storage consideration:** Multiple `node_modules` directories consume disk space. A typical Next.js project uses approximately 400-600MB per worktree. For three active worktrees, that's 1.2-1.8GB. Modern development machines handle this without performance issues.

A Diagram showing the Git Worktree structure

## Launching Multiple Claude Sessions

Open separate terminal sessions for each worktree:

**Terminal 1:**

```
cd ~/myproject
claude
```

**Terminal 2:**

```
cd ~/my
claude
```

**Terminal 3:**

```
cd ~/myproject-bug-session
claude
```

Each Claude instance operates independently with full project context in its respective branch.

## What Tasks Work in Parallel

**Not all development work suits parallel execution. Use this decision framework:**

**Ideal for parallel worktrees:**

- Independent features *(adding OAuth provider while building dashboard analytics)*
- Concurrent bug fixes *(fixing production API timeout while addressing UI rendering issue)*
- Experimental implementations *(testing Next.js 14 App Router vs. Pages Router simultaneously)*
- Code generation + review *(one instance writes, another audits)*
- Isolated refactoring *(updating one microservice while maintaining another)*

**Avoid for parallel worktrees:**

- Tasks modifying shared utilities or core libraries
- Database schema migrations affecting multiple features
- Major architectural refactors requiring system-wide awareness
- Changes to shared configuration or build systems

**The key criterion:** ***file-level independence.*** If two tasks modify the same files, they should not

## The Code + Review Pattern

**Anthropic's engineering team uses a two-instance pattern for critical features:**

```
# Terminal 1: Implementation
cd ~/myproject-auth-refactor
claude "Implement JWT refresh token rotation with Redis backing"
```

```
# Terminal 2: Security audit
cd ~/myproject-auth-review  # Separate worktree on same branch
claude "Audit the authentication implementation for security vulnerabilities,
focusing on token storage, rotation logic, and race conditions"
```

This pattern catches issues before human code review.

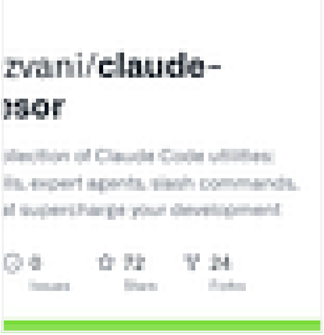
Claude Code — Helpfull Slash Commands /review or Custom Slash Commands /code-review

You can do ce project on Github: *(Any suggestions or ideas, please below in the comments or open an issue on github. Suggestions are always welcome ;))*

**GitHub - alirezarezvani/claude-code-tresor: A world-class collection of Claude Code utilities...**

A world-class collection of Claude Code utilities: autonomous skills, expert agents, slash commands, and prompts that...

github.com



. . .

## Managing Context Across Sessions

Each Claude Code session starts with zero context about your codebase, your architecture decisions, or your coding patterns. Without deliberate context management, you'll discover the same problems three times and implement three different solutions.

I learned this during a weekend sprint on a client project. Three Claude Code instances, three features, all needing caching. Each discovered and solved the caching problem independently. When I merged, I had three incompatible caching implementations. The debugging session took longer than sequential development would have.

The solution is systematic documentation that all Claude Code instances can access.

**Mastering Claude Code: A 7-Step Guide to Building AI-Powered Projects with Context Engineering**

From Chaos to Code: How I Reduced Development Time by 70% Using Claude Code's Hidden Power

alirezarezvani.medium.com

## Creating a Shared Knowledge Base

Create a project knowledge base in your repository root:

```
# CLAUD

## Arch
- Next.js 14 App Router
- PostgreSQL 15 via Prisma 5.x
- tRPC for type-safe API layer
- Redis 7 for session/cache management
## Essential Patterns
- Authentication: Middleware in /lib/auth.ts
- Database: Prisma Client singleton in /lib/db.ts
- Validation: Zod schemas in /lib/validations/*
- Testing: Jest + React Testing Library
## Coding Standards
- TypeScript strict mode enabled
- Database queries require transactions for writes
- API routes require authentication middleware
- Unit tests required for business logic
## Current Active Branches
- feature/auth-v2: JWT refresh token implementation
- hotfix/session-timeout: Session expiry bug fix
- feature/api-optimization: Response time improvements
## Critical Constraints
- No localStorage for tokens (httpOnly cookies only)
- All user input must pass Zod validation
- Rate limiting configured in middleware-do not duplicate
```

Git worktrees automatically share files from the parent repository. Every Claude instance can read this file. Reference it explicitly:

```
claude "Review CLAUDE.md for authentication patterns, then implement
refresh token rotation following the documented approach"
```

## Worktree-Specific Context

Each worktree maintains its own task context. Create `.llm/state.md`:

```
# Feature: JWT Refresh Token Rotation

## Objective
Implement secure token rotation to prevent token theft
## Technical Approach
- Store refresh tokens in Redis with user_id as key
- Tokens expire after 7 days
- Implement token family tracking for breach detection
- Handl
## Prog
- [x] F
- [x] Refresh token data model
- [ ] Token rotation logic
- [ ] Concurrent request handling
- [ ] Integration tests
## Key Decisions
- Redis instead of database for performance (50ms vs 150ms)
- Token families for breach detection
- 7-day expiry per security review
```

This maintains focus without cross-contamination from other active development streams.

. . .

## The Merge Strategy That Actually Works

Parallel development is productive until you need to merge everything back together. Here's the production-tested merge strategy.

### Daily Integration Ritual

**Merge main into feature branches every morning:**

```
# From feature worktree
cd ~/myproject-auth-refactor

git fetch origin
git merge origin/main
# Resolve conflicts immediately while they're small
npm test
npm run type-check
```

**Why daily matters:** Small, frequent merges take minutes. Weekly merges can take hours. The difference compounds across sprint cycles.

### Feature Completion Workflow

**When your work is ready for integration:**

```
# 1. Co
cd ~/my
git sta
git add .
git commit -m "feat(auth): implement JWT refresh token rotation"

# 2. Final synchronization
git fetch origin
git rebase origin/main
# 3. Validation
npm test
npm run lint
npm run type-check
npm run build
# 4. Push to remote
git push -u origin feature/auth-v2
# 5. Open pull request
# 6. Post-merge cleanup
cd ~/myproject
git pull origin main
git worktree remove ~/myproject-auth-refactor
git branch -d feature/auth-v2
```

**Reality check:** Parallel development can generate more merge conflicts than sequential work. The advantage is throughput, not conflict avoidance. Select truly independent tasks and maintain daily integration.

. . .

### What Actually Happened When We Deployed This

The first week was slower, not faster. Setup overhead, mental model shifts, inevitable mistakes — working in the wrong worktree, forgetting which terminal was which — felt like regression.

On day three, one team member wanted to abandon the approach entirely. We nearly reverted to standard workflows.

Around day ten, something shifted. We stopped thinking in branches and started thinking in workspaces. The cognitive overhead disappeared. The workflow became natural.

After two months of consistent usage, we could measure concrete changes: features per sprint i nd most importantl

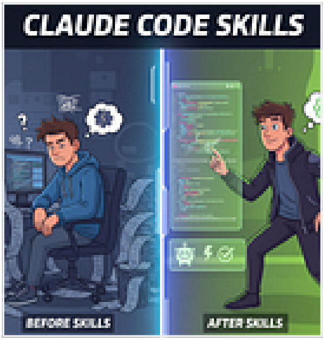
**The counterintuitive finding from the METR study:** Experienced developers using AI tools were nineteen percent slower on individual tasks but sixty-nine percent continued using them anyway. Why? Because task completion time in isolation doesn't capture the value of preserved flow states and eliminated context switching.

Our experience validated this: even when individual task times stayed similar, overall team throughput improved because multiple workstreams progressed simultaneously.

**Master Claude Code "Skills" Tool to transform repetitive AI prompts into permanent, executable...**

Discover how the Anthropic's new tool for Claude Code called "Skills" transform AI Coding assistant from a generic...

medium.com



## Task Selection Framework

We developed this decision framework through production experience:

**Create a worktree when:**

- Task requires more than thirty minutes of focused work
- Task needs distinct context from current work
- Urgent task arrives but current work shouldn't be interrupted
- Experimenting with multiple implementation approaches
- Running code generation + review pattern

**Skip the worktree when:**

- Task completes in under fifteen minutes
- Task directly relates to current work
- Task requires system-wide changes
- Working on major refactor affecting multiple areas

## Common Problems You'll Hit

### Problem 1: Worktree Accumulation

**What happened:** Created worktrees for small tasks that resolved quickly. Forgot cleanup. Ended up with seven worktrees, three abandoned, all consuming disk space and mental overhead.

**Solution:** *Weekly cleanup ritual:*

```
git worktree list
# Review each worktree
git worktree remove ~/myproject-old-experiment
git worktree prune
```

**Policy:** If worktree is inactive for three days, evaluate whether it's still needed.

### Problem 2: Configuration Drift

**What happened:** Each worktree had slightly different environment files. Tests passed in worktree A, failed in worktree B. Debugging took half a day because we didn't realize configuration was the issue.

**Solution:** *Symlink all environment files from single source:*

```
# From each worktree
rm .env .env.local
ln -s ~/myproject/.env .env
ln -s ~/myproject/.env.local .env.local
```

Single source of truth eliminates drift.

### Problem 3: Infrequent Main Integration

**What happened:** Let feature branch diverge for two weeks. Merge required extensive conflict resolution across twenty-three files.

**Solution:** *Daily standup includes “sync worktrees with main.”* Takes five to ten minutes, p

**Problem 4:**

**What happened:** Merged AI-generated code with surface-level review. Introduced subtle race condition in session handling that caused intermittent production issues.

**Solution:** ***Mandatory two-instance review pattern.*** One Claude generates code, second Claude audits for correctness, performance, and security. Catches issues before human review.

• • •

## Why Traditional Productivity Metrics Miss the Point

Traditional productivity metrics measure the wrong things for parallel development.

The METR study found experienced developers were slower with AI tools like Claude Code or any other Coding assistants *(e.g. OpenAI Codex CLI or Google Gemini CLI)* but continued using them. This paradox reveals that task completion time in isolation misses critical factors.

**What traditional metrics miss:**

- Cognitive load *(mental overhead of context switching)*
- Flow state preservation *(uninterrupted deep work periods)*
- Parallel throughput *(multiple workstreams progressing simultaneously)*
- Context preservation *(no re-explanation penalty between sessions)*

**What actually matters:**

- Can urgent bugs get fixed without interrupting feature work?
- Can we experiment with multiple approaches without abandoning progress?
- Does switching between tasks require mental context rebuilding?
- How much time is lost bringing Claude Code up to speed on project context?

When you optimize for these factors, parallel worktrees show clear advantages — even if individual task times don’t decrease dramatically.

Research analyzing three hundred engineers over twelve months found statistically significant *ours, p =* *0.0018)*. Th g.

**Claude AI and Claude Code Skills: Teaching AI to Think Like Your Best Engineer**

medium.com

. . .

## The Contrarian Take: When Sequential Works Better

Parallel development adds complexity. If that complexity doesn't buy meaningful throughput improvement, it's waste.

**Stick with sequential work when:**

- Working alone on single feature with clear scope
- Tasks are highly interdependent
- Codebase is small (under 10,000 lines)
- Team is still learning Claude Code basics
- Major refactoring requires system-wide awareness
- Merge conflicts would be expensive (shared utilities, database schemas)

**The honest assessment:** For senior engineers working on complex, interconnected systems, parallel execution sometimes increases total time due to integration overhead.

**The pattern works best for teams with:**

- Clear feature boundaries
- Good modular architecture
- Multiple concurrent priorities
- High context-switching costs

If you don't have these conditions, sequential development might be more efficient.

The incide because their architecture supports it. Their microservices structure and clear service boundaries make parallel work natural. Your architecture might not.

Evaluate your specific context before investing in changes.

. . .

## Your Implementation Roadmap

**Initial setup (30 minutes):**

- [ ] Select two independent tasks from current sprint
- [ ] Create worktrees for each task
- [ ] Symlink environment files
- [ ] Install dependencies in each worktree
- [ ] Launch Claude in separate terminal per worktree
- [ ] Note subjective experience differences

**First week:**

- [ ] Create CLAUDE.md with architecture and patterns
- [ ] Create per-worktree state files
- [ ] Implement daily main integration ritual
- [ ] Try code generation + review pattern
- [ ] Document which tasks benefit from parallel execution

**First month:**

- [ ] Expand to three to four worktrees for sprint work
- [ ] Identify task patterns that work well in parallel
- [ ] Create automation scripts for worktree setup
- [ ] Share learnings with team

- [ ] Measure cycle time changes

**Claude Code Plugins: The 30-Second Setup That Turned Our Junior Dev Into a Deployment Expert**

What took engineers weeks to build now installs in one command. Here's how AI coding finally became shareable — and why...

medium.com

· · ·

## Frequently Asked Questions

**Q: Do git worktrees work with GitHub, GitLab, and Bitbucket?**

Yes, worktrees are a standard Git feature (available since Git 2.5). They work with any Git-based platform. The remote repository sees normal branches — it doesn't know or care that you're using worktrees locally.

**Q: How much disk space do multiple worktrees actually consume?**

Each worktree only duplicates working files, not Git history. JavaScript projects typically use 400–600MB per worktree (mostly `node_modules`). Python projects are smaller, usually 50-200MB depending on dependencies. Three worktrees for a Next.js project: approximately 1.5-2GB total.

**Q: Can I use this with Cursor, GitHub Copilot, or other AI tools?**

Absolutely. The worktrees pattern works with any AI coding tool: Cursor, GitHub Copilot CLI, Aider, or any other terminal-based assistant. The isolation benefits apply universally.

**Q: What happens if I delete a worktree with uncommitted changes?**

Git prevents deleting worktrees with uncommitted changes by default. You'll get a warning. Use `git worktree remove --force` only if you're certain you want to discard changes.

**Q: How do I share this pattern with my team?**

Document the setup in your repository's README. Include worktree creation commands, environment setup steps, and daily integration practices. Most teams adopt the pattern within one to two weeks once they see one developer using it effectively.

**Q: Does this work with monorepos?**

Yes, but with considerations. In monorepos, create worktrees for specific packages or services

independe

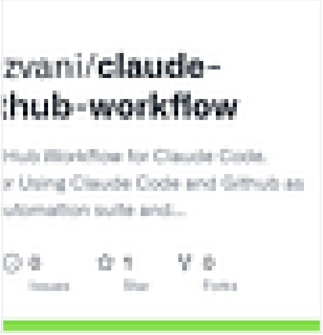
**Boost your GitHub productivity with this beginner friendly, one-click installation automation framework for Claude Code projects.**
This free and open source toolkit brings end-to-end automation to your repository, including smart GitHub Actions workflows, automated issue and project management, streamlined PR and code review routines, built-in security scans, and more.

If you want a fast, reliable way to scale your development workflow and strengthen CI/CD without extra setup, this framework gives you a ready-to-use automation layer in minutes. Perfect for developers, teams, and anyone aiming for cleaner, safer, and more efficient GitHub operations.

**GitHub - alirezarezvani/claude-code-github-workflow: World-Class GitHub Workflow for Claude Code...**

World-Class GitHub Workflow for Claude Code. This Blueprint for Using Claude Code and Github as your Workflow...

github.com


zvani/claude-
hub-workflow

. . .

## The Bottom Line

Git worktrees combined with parallel Claude Code instances solve a structural problem: context switching creates unavoidable productivity loss when constrained to single-workspace development.

Traditional git workflows — one workspace, constant branch switching — force context loss as a structural requirement. Every branch switch means lost mental context. Every Claude restart means re-explaining your codebase.

Worktrees remove that structural constraint. Multiple isolated workspaces. Multiple Claude instances. Each maintains full context. Each progresses independently. Zero switching overhead. Zero context loss.

**This isn't about working faster. It's about removing unnecessary barriers.**

The patter [illegible] ments. Results vary based on team structure, architecture quality, and task independence. The METR study showed nineteen percent slowdown for experienced developers on individual tasks, while production teams like incident.io report substantial cycle time improvements.

The difference? Context. Task selection. Architecture that supports parallel work.

**Evaluate whether your specific situation benefits from parallel execution:**

Ask yourself: Do we have multiple concurrent priorities? Clear feature boundaries? High context-switching costs? If yes, start with two worktrees this week. Run the experiment. Measure what actually improves.

If no, stick with sequential work. Sometimes the simplest solution is the right solution.

The goal isn't adopting every new pattern. It's identifying which structural problems actually constrain your team, then implementing solutions that address root causes rather than symptoms.

Context switching is a structural problem. Worktrees are a structural solution. Whether that solution fits your situation — that's what the next week will tell you.

**Claude Code v2.0.30: Full Guide of what is New? Production Readiness Edition**

How Anthropic's latest version of Claude Code 2.0.30 transforms reliability, security, everyday developer experience…

alirezarezvani.medium.com

. . .

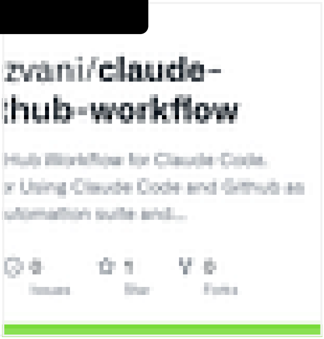
**Boost your GitHub productivity with this beginner friendly, one-click installation automation framework for Claude Code projects.**
This free and open source toolkit brings end-to-end automation to your repository, including smart GitHub Actions workflows, automated issue and project

management, streamlined PR and code review routines, built-in security scans, and more.

**GitHub - alirezarezvani/claude-code-github-workflow: World-Class GitHub Workflow for Claude Code...**

World-Class GitHub Workflow for Claude Code. This Blueprint for Using Claude Code and Github as your Workflow...

github.com



If you want a fast, reliable way to scale your development workflow and strengthen CI/CD without extra setup, this framework gives you a ready-to-use automation layer in minutes. Perfect for developers, teams, and anyone aiming for cleaner, safer, and more efficient GitHub operations.

## Join the Conversation

**What challenges have you encountered with parallel development?** Share specific experiences — especially failures or unexpected problems. Those insights help everyone evaluate whether this pattern fits their situation.

If this guide provided value, engagement helps others discover it. Medium’s algorithm surfaces content based on reader interaction.

**Follow for production-tested development patterns and architecture decisions** — no hype, just engineering reality from teams shipping to production.

## Further Reading

- Anthropic: Claude Code Best Practices — Official engineering patterns
- Git Worktree Documentation — Complete technical reference
- METR: Measuring AI Impact on Developer Productivity — Controlled study results
- Claude Code Common Workflows — Official implementation patterns

Welcome back. You are signed into your member account **bg••••@jaxondigital.com**.

**About This** *parallel developmen real environments. Metrics cited are from published research or verified production deployments.*

**Connect:** *For questions about implementing git worktrees in your specific context, reach out in the comments. I respond to technical questions and share additional patterns that didn't fit this guide. Soon I will provide the Github Open Source Repository of this Article.*

. . .

✨ *Thanks for reading! If you'd like more practical insights on AI and tech, hit* ***subscribe*** *to stay updated.*

*I'd also love to hear your thoughts — drop a comment with your ideas, questions, or even the kind of topics you'd enjoy seeing here next. Your input really helps shape the direction of this channel.*

### About the Author

*Me, Alireza Rezvani work as a CTO @ an HealthTech startup in Berlin and architect AI development systems for my engineering and product teams. I write about turning individual expertise into collective infrastructure through practical automation.*

**Connect:** Website | LinkedIn
**Read more:** Medium (Reza Rezvani)

**Explore my other open source projects:** GitHub

Git Worktree | Claude Code | Context Switching | Coding | DevOps

Following

## Written by Reza Rezvani

1.1K followers · 77 following

As CTO of a B des in tech, I drive innovati

Welcome back. You are signed into your member account **bg••••@jaxondigital.com**.

## No responses yet

Bgerby

What are your thoughts?

## More from Reza Rezvani

In nginity by Reza Rezvani

### How Cursor and Claude Code Plugins Turned Me Into a 20x Developer – The Agentic Coding Setup That...

My Backgro o wrote this?”

Welcome back. You are signed into your member account **bg••••@jaxondigital.com**.

✦ Oct 14

Reza Rezvani

## Gemini CLI: What Happened When I Replaced My IDE With a Free AI Terminal Agent for 30 Days

I gave Google’s new Gemini CLI full access to my development workflow and tested it on real production code. Here’s what actually worked…

✦ Oct 10 👏 20

Reza Rezv

Welcome back. You are signed into your member account **bg••••@jaxondigital.com**.

## “7 Steps” ng (Part 1): The Foundation of...

Learn how to stop Claude Code from rewriting your architecture with vague prompts. This guide introduces Spec-Driven Development...

✦ Sep 17 62 2

Reza Rezvani

## Master Claude Agent SDK: A 5-Step Integration Guide to Cut Development Time 70% with...

Stop building custom agent loops from scratch. Follow this battle-tested integration to deploy secure, scalable AI agents in Claude Code

✦ 2d ago 5

See all from Reza Rezvani

## Recommended from Medium

Barnacle Goose

### How GPT-5-Codex Compares to Claude Sonnet 4.5

The fall of 2025 was marked by the arrival of two contenders from the industry’s leading AI labs. On September 15, OpenAI launched...

Oct 17 27 1

The Atom

Welcome back. You are signed into your member account **bg••••@jaxondigital.com**.

## You Have re—It Redefines Insane

I built a working expense tracker in eight minutes while my coffee was still hot, and it remembered every receipt when I closed my laptop...

✦ Oct 26 252 17

Aakash Gupta

## Amazon Just Fired 30,000 People After $60B in Profits. Here's What They're Not Telling You.

Amazon is laying off 30,000 workers.

5d ago 182 15

Welcome back. You are signed into your member account **bg••••@jaxondigital.com**.

Daniel Avila

## Claude Code Learning Path: a practical guide to getting started

After spending months diving deep into Claude Code, I wanted to share the learning path that worked for me. This isn’t from official…

Oct 30 122

ZIRU

## Why Spec-Driven Development Made Me 5x More Productive

And How GitHub Spec Kit Changed Everything

Oct 16

Welcome back. You are signed into your member account **bg••••@jaxondigital.com**.

Reza Rezvani

**Master Claude Agent SDK: A 5-Step Integration Guide to Cut Development Time 70% with...**

Stop building custom agent loops from scratch. Follow this battle-tested integration to deploy secure, scalable AI agents in Claude Code

2d ago 5

See more recommendations