คู่มือสำหรับ Developer ทั่วไป
ในการเขียนโค้ดแบบ Agentic ด้วย Claude Code

Agentic Coding ด้วย Claude Code

1. ใช้ Claude Code เป็นคู่หู
สร้างระบบอัตโนมัติและ
เวิร์กโฟลว์แบบ Agentic ได้ดั่งใจ

2. สร้างและต่อยอด แอป Next.js
พัฒนาแอป Next.js
ต่อยอดได้เรื่อยๆ
โดยมี AI ช่วยทุกขั้นตอน

3. สเกลงานไม่ลด คุณภาพโค้ด
ขยายการพัฒนาด้วย AI
พร้อมรักษาคุณภาพโค้ดไว้ครบ

Agentic Coding with Claude Code

คู่มือ Agentic Coding with Claude Code สำหรับ developer ในการทำงานประจำวัน

อ้างอิงจาก *Agentic Coding with Claude Code* โดย Eden Marco

เรียบเรียงและจัดทำโดย **Agentic Stack**

หมายเหตุเกี่ยวกับฉบับเรียบเรียงภาษาไทย

เอกสารนี้เป็นฉบับเรียบเรียงและดัดแปลงภาษาไทยโดยคณะผู้จัดทำ **Agentic Stack** อ้างอิงจาก *Agentic Coding with Claude Code* โดย Eden Marco จัดทำขึ้นเพื่อการศึกษาและการใช้งานส่วนบุคคล มิใช่ฉบับแปลหรือฉบับจัดพิมพ์อย่างเป็นทางการ

ศัพท์เทคนิคด้านคอมพิวเตอร์ AI และชื่อ feature ของ Claude Code คงรูป English แบบ canonical ส่วนคำกริยาและคำเชื่อมเรียบเรียงเป็นภาษาไทย เพื่อรักษาความหมายทางเทคนิคและให้อ่านได้อย่างเป็นธรรมชาติ

ฉบับภาษาไทย

สารบัญ

Agentic Coding with Claude Code

คำนำ

generative AI และ coding agent กำลังเปลี่ยนแปลงวิธีสร้าง software อย่างรวดเร็ว tool อย่าง Claude Code, Cursor และ LangChain deep agents ไม่ได้จำกัดอยู่เพียง interaction แบบ prompt ง่าย ๆ อีกต่อไป tool เหล่านี้ทำงานครอบคลุมทั้ง codebase ผสานรวมกับ tool ภายนอก และเรียกใช้ workflow แบบ multi-step ที่ทำงานระยะยาว เมื่อระบบเหล่านี้มีความสามารถมากขึ้น การเข้าใจวิธีทำงานและวิธีควบคุมระบบจึงกลายเป็นสิ่งจำเป็น

หนังสือเล่มนี้มุ่งสร้างความเข้าใจที่แข็งแกร่งและนำไปใช้ได้จริงเกี่ยวกับ **Claude Code**, agentic workflow และ concept เบื้องหลังที่ทำให้ coding agent สมัยใหม่ทำงานได้อย่างมีประสิทธิภาพ theme หลักตลอดทั้งเล่มคือ Context Engineering ซึ่งอธิบายว่า agent สร้าง จัดการ และใช้ context อย่างไรในระหว่างที่ให้เหตุผลและลงมือทำ เมื่อ agent ซับซ้อนและขับเคลื่อนด้วย tool มากขึ้น การจัดการ context อย่างเหมาะสมจะส่งผลโดยตรงต่อ performance ต้นทุน reliability และความสม่ำเสมอ

ในหนังสือเล่มนี้ คุณจะได้เรียนรู้ว่า Claude Code โต้ตอบกับ codebase อย่างไร วิธีกำหนดค่าสำหรับการใช้งานประจำวัน และวิธีต่อขยายด้วย **Model Context Protocol** (MCP) คุณจะได้สำรวจ automation ด้วย GitHub Actions planning อย่างมีโครงสร้าง workflow แบบ multi-agent subagent Output style และ Agent Skills หนังสือยังพิจารณาหัวข้อเชิง architecture ที่ลึกยิ่งขึ้น รวมถึงวิธีทำงานของ deep agent และวิธีนำไปใช้ workflow แบบ long-horizon ในทางปฏิบัติ

แทนที่จะมุ่งเน้นเพียงการใช้งานระดับพื้นผิว หนังสือเล่มนี้มีเป้าหมายช่วยให้คุณเข้าใจว่าระบบเหล่านี้ได้รับ design อย่างไร และเหตุใดจึงมีพฤติกรรมเช่นนั้น ด้วยการผสานตัวอย่างเชิงปฏิบัติเข้ากับ insight ด้าน architecture ที่ลึกกว่า เป้าหมายคือมอบทั้ง skill สำหรับใช้งาน coding agent สมัยใหม่อย่างมีประสิทธิภาพ และความเข้าใจที่จำเป็นต่อการปรับตัวในขณะที่วงการนี้พัฒนาต่อไป

หนังสือเล่มนี้เหมาะกับใคร

หนังสือเล่มนี้เหมาะสำหรับ developer และ engineer ที่ต้องการก้าวพ้น AI แบบ chat-based และสร้าง agentic workflow สำหรับ development แบบอัตโนมัติด้วย Claude Code และ MCP หนังสือเขียนขึ้นสำหรับ application developer ที่มีประสบการณ์ด้าน software engineering มาก่อน และต้องการผสานรวม AI agent แบบ context-aware เข้ากับ terminal และ IDE ของตน ทำให้งาน coding ที่ซับซ้อนเป็นอัตโนมัติ และออกแบบระบบ multi-agent ที่ขยายขนาดได้

นอกจากนี้ หนังสือยังเป็นประโยชน์ต่อ AI engineer และผู้ปฏิบัติงานด้าน generative AI ที่ทำงานใกล้ชิดกับ development workflow สมัยใหม่

เพื่อให้ได้ประโยชน์สูงสุดจากหนังสือเล่มนี้ คุณควรมีประสบการณ์เขียนและแก้จุดบกพร่อง code ด้วย Python หรือ TypeScript คุ้นเคยกับการทำงานด้วย Git และ development environment ตลอดจนเข้าใจ concept หลักของ generative AI เช่น LLM, RAG และ agent หนังสือเล่มนี้ไม่ใช่หนังสือระดับเริ่มต้น และตั้งอยู่บนสมมติฐานว่าผู้อ่านมีประสบการณ์ด้าน software

development มาก่อน

เนื้อหาที่หนังสือเล่มนี้ครอบคลุม

บทที่ 1, *Context Engineering* แนะนำ Context Engineering และอธิบายว่าเหตุใดจึงมีความสำคัญต่อการสร้าง AI agent สมัยใหม่ บทนี้ครอบคลุมว่า context วิวัฒนาการมาจาก prompt engineering อย่างไร เติบโตขึ้นในระบบที่ซับซ้อนอย่างไร และการจัดการ context ที่ไม่มีประสิทธิภาพส่งผลต่อ performance และต้นทุนอย่างไร นอกจากนี้ ยังพิจารณา system prompt และการจัดการ context เชิงปฏิบัติด้วย Claude Code

บทที่ 2, *แก่นสำคัญของ Claude Code* นำเสนอข้อมูลเบื้องต้นเชิงปฏิบัติเกี่ยวกับ Claude Code และอธิบายว่า Claude Code โต้ตอบกับ codebase อย่างไร บทนี้ครอบคลุมการเริ่มต้น project file `CLAUDE.md` permission และการจัดการ context ทั้งยังแนะนำ MCP, design แบบ spec-driven และเริ่มสร้าง HookHub project ที่จะใช้ตลอดทั้งเล่ม

บทที่ 3, *เริ่มต้นใช้งาน Claude Code – สำรวจ command สำคัญ* มุ่งเน้นการกำหนดค่า Claude Code สำหรับใช้งานประจำวัน บทนี้ครอบคลุมราคาและ authentication command configuration ระดับ user และระดับ project ตลอดจน integration กับ Cursor นอกจากนี้ ยังสำรวจ hook checkpointing Skills และบทบาทของ Language Server Protocol

บทที่ 4, *ขยายขีดความสามารถของ Claude Code ด้วย MCP server และ plugin* อธิบาย Model Context Protocol และวิธีที่ protocol นี้สร้างมาตรฐานให้ integration ระหว่าง AI application กับ tool ภายนอก บทนี้ครอบคลุม architecture ของ MCP ซึ่งรวมถึง host client และ server พร้อมสาธิตวิธีกำหนดค่า MCP server ทั้งแบบ local และ remote นอกจากนี้ ยังกล่าวถึงการจัดการ context ข้อพิจารณาด้าน performance และการต่อขยายด้วย plugin

บทที่ 5, *ทำให้ development workflow เป็นอัตโนมัติด้วย Claude Code และ GitHub* สำรวจ integration ระหว่าง Claude Code กับ GitHub บทนี้ครอบคลุม configuration ของ repository automation สำหรับ pull request และ issue รวมถึงการใช้ GitHub Actions เพื่อเรียกใช้ workflow ทั้งยังอธิบายว่า YAML workflow file นิยาม automation แบบ event-driven อย่างไร

บทที่ 6, *planning ด้วย Claude Code และ workflow แบบ multi-agent* พิจารณา agentic workflow ที่มีโครงสร้างผ่าน planning และการทำงานแบบ multi-agent ที่ประสานกัน บทนี้อธิบายว่า spec-driven development ช่วยเพิ่มความสามารถในการคาดการณ์ได้อย่างไร และ agent หลายตัวสามารถร่วมงานกันภายใน codebase เดียวกันได้อย่างไร

บทที่ 7, *ทำงานกับ subagent ของ Claude Code* แนะนำ subagent ของ Claude Code และอธิบายว่า subagent ทำให้เกิด workflow ที่มีโครงสร้างและแยกจากกันได้อย่างไร บทนี้ครอบคลุม subagent configuration แบบ custom flow ของ context และการทำงานพร้อมกันด้วย pattern Infinite Agentic Loop

บทที่ 8, *สร้างและปรับแต่ง Output style* พิจารณาว่า Output style กำหนดรูปแบบ response ของ Claude Code อย่างไร บทนี้ครอบคลุมการสร้างและกำหนด scope ให้ custom style format ที่มีโครงสร้างอย่าง YAML และการทำให้พฤติกรรมภายในนิยามของ style เป็นอัตโนมัติ ทั้งยังอธิบายวิธีจัดการบทบาทและ configuration ของ session

บทที่ 9, *ทำความเข้าใจ Agent Skills* สำรวจ Agent Skills ในฐานะกลไกสำหรับต่อขยายความสามารถของ AI agent บทนี้ครอบคลุม concept พื้นฐาน การใช้งานจริงใน Claude Code และ LangChain DeepAgent รวมถึง flow ภายในของ context เมื่อมีการ invoke skill ก่อนปิดท้ายด้วยตัวอย่าง implementation จริง

บทที่ 10, *ใช้งาน Claude Code Desktop* อธิบายวิธีใช้ Claude Code ภายใน desktop application บทนี้ครอบคลุมการสลับระหว่าง local mode กับ cloud mode การเรียกใช้ agent แบบ parallel ด้วย Git worktree และ orchestration feature branch ข้าม environment

บทที่ 11, *ทำความเข้าใจ deep agent* พิจารณา deep agent และบทบาทในการเรียกใช้ task แบบ long-horizon บทนี้นิยาม characteristic ของ deep agent และวิเคราะห์ agent harness ของ LangChain deep agents ซึ่งรวมถึง architecture และ model การทำงาน

บทที่ 12, *Claude Code รุ่นปัจจุบัน: จาก autonomous workflow สู่ production-ready operations* เรียบเรียง capability ที่เพิ่มใน Claude Code ช่วง Week 13 ถึง Week 28 ปี 2026 โดยคัดเฉพาะส่วนที่ยังไม่มีในบทเดิม บทนี้ครอบคลุม Claude Sonnet 5, Agent view, background และ nested subagent, `/goal`, Dynamic workflows, browser และ Computer use, Artifacts, Auto mode, permission rules, code review ตลอดจน diagnostic และ operation command รุ่นปัจจุบัน

เพื่อให้ได้ประโยชน์สูงสุดจากหนังสือเล่มนี้

หนังสือเล่มนี้ตั้งอยู่บนสมมติฐานว่าผู้อ่านมีประสบการณ์ด้าน software development และ concept ของ generative AI มาก่อน ก่อนเริ่มอ่าน คุณควรคุ้นเคยกับการเขียนและแก้จุดบกพร่อง code ด้วย Python หรือ TypeScript การเรียกใช้โปรแกรมจาก terminal และการทำงานกับ codebase หนังสือคาดหวังให้คุณมีความรู้พื้นฐานเกี่ยวกับ Git เช่น การ clone repository และการสร้าง commit การเปลี่ยนแปลง รวมทั้งควรเข้าใจ virtual environment และ environment variable ด้วย

คุณจำเป็นต้องคุ้นเคยกับ **large language model** (**LLM**) และ concept หลักอย่าง agent RAG และ ReAct คุณควรเคยโต้ตอบกับ LLM และสร้าง agent อย่างง่ายมาแล้วอย่างน้อยหนึ่งตัว หนังสือเล่มนี้ไม่ครอบคลุมการเขียนโปรแกรมระดับเริ่มต้นหรือหัวข้อเบื้องต้นเกี่ยวกับ generative AI

หากต้องการทำตามตัวอย่างแบบลงมือปฏิบัติ คุณจำเป็นต้องมี development environment ที่ใช้งานได้ พร้อม Node.js และตั้งค่า Next.js แบบง่ายตามที่แนะนำไว้ในบทแรก ๆ คุณยังต้องเข้าถึง Claude Code พร้อม configuration ด้าน authentication และราคาที่เหมาะสม บางบทจำเป็นต้องติดตั้งและกำหนดค่า GitHub CLI ทำงานกับ GitHub repository และใช้ GitHub Actions ส่วนท้าย ๆ จะเกี่ยวข้องกับการกำหนดค่า MCP server ทั้งแบบ local และ remote รวมถึงการเรียกใช้ Claude Code ภายใน Cursor และ Claude Desktop application

ขอแนะนำให้คุณทำตามตัวอย่างทีละขั้น และทดลองใช้ configuration เมื่อมีการแนะนำในแต่ละช่วง หนังสือสร้างเนื้อหาต่อยอดขึ้นตามลำดับ และหลายบทอาศัย project และการตั้งค่าที่จัดเตรียมไว้ก่อนหน้านี้ รวมถึง HookHub project

ตัวอย่างและ screenshot ในหนังสือเล่มนี้ได้รับการจัดลำดับและเรียบเรียงใหม่เพื่ออธิบาย workflow เชิงเทคนิคอย่างต่อเนื่อง โดยใช้ repository สำหรับการเรียนรู้เป็นกรณีศึกษา

รูปแบบที่ใช้ในหนังสือ

หนังสือเล่มนี้ใช้รูปแบบข้อความหลายประเภท

`CodeInText` : ใช้ระบุคำที่เป็น code ภายในข้อความ ชื่อตาราง database ชื่อ folder ชื่อ file นามสกุล file path URL จำลอง user input และ handle บน X/Twitter ตัวอย่างเช่น: “ทำได้โดยเริ่มต้น file `CLAUDE.md` เพื่อให้เพิ่มเข้าไปใน context ของ project ได้”

code block จะแสดงดังต่อไปนี้:

```
add_context() {
  local context_ref="$1"
  grep -qxF "$context_ref" "$CLAUDE_MD" || echo "$context_ref" >>
    "$CLAUDE_MD"
}
```

input หรือ output ของ command line จะแสดงดังต่อไปนี้:

```
npx create-next-app@latest
```

ตัวหนา: ใช้ระบุคำศัพท์ใหม่ คำสำคัญ หรือคำที่คุณเห็นบนหน้าจอ เช่น คำในเมนูหรือ dialog box จะแสดงในข้อความด้วยรูปแบบนี้ ตัวอย่างเช่น: “ตอนนี้ให้เลือก **Yes (ใช่)** และเลือกไม่ให้ระบบถามอีกใน session นี้”

> คำเตือนหรือหมายเหตุสำคัญจะแสดงในรูปแบบนี้

> เคล็ดลับและเทคนิคจะแสดงในรูปแบบนี้

ส่วนที่ 1

รากฐานของ Context Engineering และ Claude Code

ใน *ส่วนที่ 1* ของหนังสือเล่มนี้ คุณจะสร้างความเข้าใจที่ชัดเจนและนำไปใช้ได้จริงเกี่ยวกับ Context Engineering รวมถึงเหตุผลที่เรื่องนี้มีความสำคัญเมื่อทำงานกับ AI coding agent สมัยใหม่ เราจะพิจารณาว่าการเปลี่ยนผ่านจาก prompt engineering แบบเรียบง่ายไปสู่การจัดการ context อย่างมีโครงสร้าง ส่งผลต่อวิธีออกแบบและใช้งาน agent อย่างไร คุณจะได้สำรวจว่า context ถูกสร้างขึ้นอย่างไร เติบโตขึ้นตามเวลาอย่างไร และการจัดการที่ไม่มีประสิทธิภาพส่งผลต่อ performance ต้นทุน และ reliability อย่างไร

ควบคู่ไปกับรากฐานเชิง concept คุณจะเริ่มลงมือทำงานกับ Claude Code โดยตรง ผ่านตัวอย่างที่มีคำแนะนำเป็นลำดับ คุณจะได้เห็นว่า Claude Code ตีความ codebase สร้าง working context และนำ context นั้นไปใช้กับงาน development จริงอย่างไร เมื่อจบส่วนนี้ คุณจะมีความเข้าใจที่มั่นคงทั้งด้านทฤษฎีและการใช้งาน Claude Code ในแต่ละวัน

ส่วนนี้ของหนังสือประกอบด้วยบทต่อไปนี้:

- *บทที่ 1, Context Engineering*
- *บทที่ 2, แก่นสำคัญของ Claude Code*
- *บทที่ 3, เริ่มต้นใช้งาน Claude Code – สำรวจ command สำคัญ*

1

Context Engineering

บทนี้แนะนำ Context Engineering และอธิบายว่าเหตุใด concept นี้จึงมีความสำคัญอย่างยิ่งต่อการสร้างและใช้งาน AI agent สมัยใหม่ แม้ระบบ AI จำนวนมากมักถูกอธิบายว่าเป็น “เพียง prompt ที่ห่อหุ้ม LLM” แต่บทนี้จะแสดงให้เห็นว่าเหตุใดมุมมองดังกล่าวจึงใช้ไม่ได้เมื่อ agent ซับซ้อนขึ้น ทำงานยาวนานขึ้น และขับเคลื่อนด้วย tool คุณจะได้เรียนรู้ว่า context มาจากที่ใด เหตุใดจึงเพิ่มขึ้นอย่างต่อเนื่อง และการจัดการ context ที่ไม่ดีนำไปสู่ performance ที่ลดลง ค่าใช้จ่ายสูงขึ้น hallucination และพฤติกรรมที่ไม่สม่ำเสมอได้อย่างไร

เป้าหมายของบทนี้คือมอบ mental model ที่ชัดเจนเกี่ยวกับ Context Engineering และแสดงวิธีนำไปใช้ในทางปฏิบัติ เราจะเริ่มด้วยการอธิบายว่า Context Engineering วิวัฒน์มาจาก prompt engineering อย่างไร จากนั้นใช้ Claude Code เป็นตัวอย่างที่เป็นรูปธรรมว่า agent สมัยใหม่เขียน เลือก บีบอัด และแยก context อย่างไร ในช่วงท้าย เราจะพิจารณา system prompt เหตุใด system prompt ยังคงสำคัญ และวิธีออกแบบอย่างมีประสิทธิผล

บทนี้จะครอบคลุมหัวข้อต่อไปนี้:

- บทนำสู่ Context Engineering
- Claude Code และปรัชญาด้าน Context Engineering
- system prompt และเหตุผลที่สำคัญ

> การอภิปรายนี้อ้างอิงข้อมูลที่เปิดเผยต่อสาธารณะ รวมถึง blog post ด้าน engineering ของ Anthropic และการวิเคราะห์จาก community เนื้อหานี้ไม่ได้เป็นถ้อยแถลงอย่างเป็นทางการจาก Anthropic

บทนำสู่ Context Engineering

ในส่วนนี้ เราจะเจาะลึกว่า Context Engineering คืออะไร หากคุณเคยทำงานกับ AI agent และอาจรวมถึง coding agent เช่น Cursor และ Claude Code หรือบางทีคุณอาจเคยพัฒนา AI agent สำหรับบริษัทหรือใช้งานเอง คุณน่าจะทราบว่าโดยพื้นฐานแล้ว ทุกอย่างล้วนลงเอยที่การส่ง prompt ไปยัง LLM พร้อมกับงานด้าน engineering จำนวนมากที่อยู่รอบ prompt นั้น

คำกล่าวที่ว่า application อย่าง Cursor และ Claude Code เป็นเพียงตัวห่อหุ้ม LLMs นั้นมีส่วนจริงอยู่บ้าง อย่างไรก็ตาม การสร้างตัวห่อหุ้มที่ดีมากต้องอาศัยทั้งความรู้เชิงลึกและงาน engineering จำนวนมาก อีกคำหนึ่งที่มักใช้เรียกระบบแวดล้อมนี้คือ *Agentic harness* ซึ่งเป็น orchestration layer รอบ LLM โดยรับผิดชอบการจัดการ tool call ควบคุม agentic loop จัดการ error บังคับใช้ guardrail และที่สำคัญที่สุดคือตัดสินใจว่าจะส่ง context ใดไปยัง model ในแต่ละขั้นตอน

ในทางปฏิบัติ งาน engineering จริงส่วนใหญ่ไม่ได้อยู่ภายในตัว LLM call แต่อยู่รอบ ๆ การเรียก model มักไม่ซับซ้อน สิ่งที่กำหนดว่า agent จะทำงานได้อย่างน่าเชื่อถือหรือไม่ คือวิธีที่ระบบแวดล้อมจัดการ state, tool, memory และ context เพราะการเรียก LLMs มาพร้อม context เสมอ และ context นี้มาจากแหล่งกับ process ต่อเนื่องที่หลากหลาย

ตัวอย่างเช่น context อาจมาจากหลายแหล่ง:

- อาจมาจาก application developer
- อาจมาจาก user
- อาจมาจาก interaction ก่อนหน้าของ user tool call หรือ data ภายนอกอื่น ๆ

ทุกวันมีการเพิ่มแหล่ง context ใหม่ และปริมาณ context ก็เพิ่มขึ้นอย่างต่อเนื่อง การส่ง context ที่ถูกต้องและเกี่ยวข้องไปยัง LLM ไม่ได้เรียบง่ายอย่างที่เราเคยคิดในยุคแรก

จาก prompt engineering สู่ Context Engineering

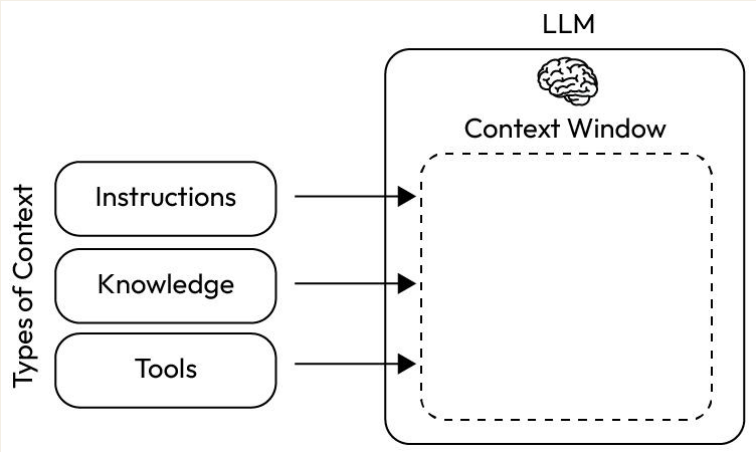
ในยุคแรก เราเชื่อว่า prompt engineering ก็เพียงพอแล้ว เราคิดว่าการเขียน prompt สวยหรูจะช่วยแก้ปัญหาและให้สิ่งที่เราต้องการได้ แต่ปัญหาคือ prompt เป็นแบบ static ขณะที่ context มีความ dynamic อย่างยิ่ง

หาก context เป็นแบบ dynamic การสร้าง context ที่ถูกต้องก็ต้องอาศัยระบบ dynamic เช่นกัน เรื่องนี้ไม่ได้มีเพียงการเขียน static prompt อีกต่อไป นี่คือเหตุผลที่เรากำลังก้าวเข้าสู่อาณาจักรของ **Context Engineering** ซึ่งเป็นวิวัฒนาการตามธรรมชาติของ prompt engineering แต่เป็น concept ที่ลึกซึ้งกว่ามาก

เหตุผลที่ context สำคัญต่อ agent

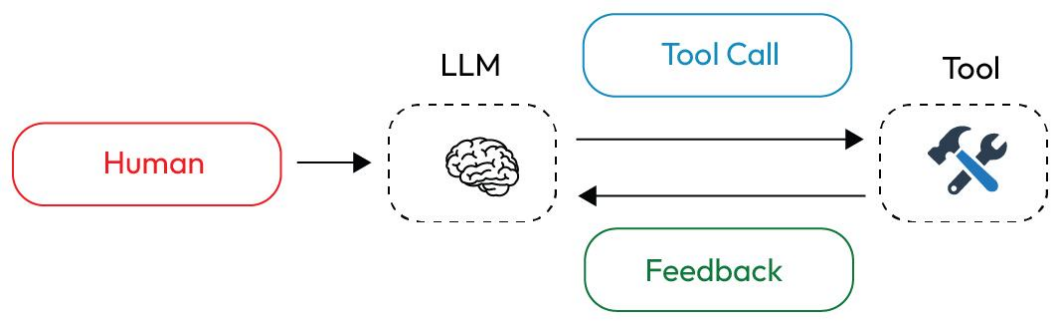
เราทุกคนรู้จักคำกล่าวที่ว่า *"ใส่ขยะเข้าไป ก็ได้ขยะออกมา"* นี่เป็นหนึ่งในสาเหตุที่พบบ่อยที่สุดซึ่งทำให้ระบบ agentic ทำงานไม่ได้ตามที่ควร เพราะระบบไม่ได้รับ context ที่ถูกต้อง

LLMs ไม่สามารถอ่านใจเราได้ เราต้องให้ข้อมูลที่ถูกต้อง และที่จริงแล้ว สิ่งที่ต้องให้ไม่ใช่เพียงข้อมูลหรือ data เสมอไป บางครั้งเราต้องมอบ tool ที่ถูกต้อง เพื่อให้ model ดึงข้อมูลอื่น ดำเนินการ และทำงานแทนเราได้



รูปที่ 1.1 – context window ของ LLM (ดัดแปลงจาก concept ที่ LangChain กล่าวถึง, www.langchain.com)

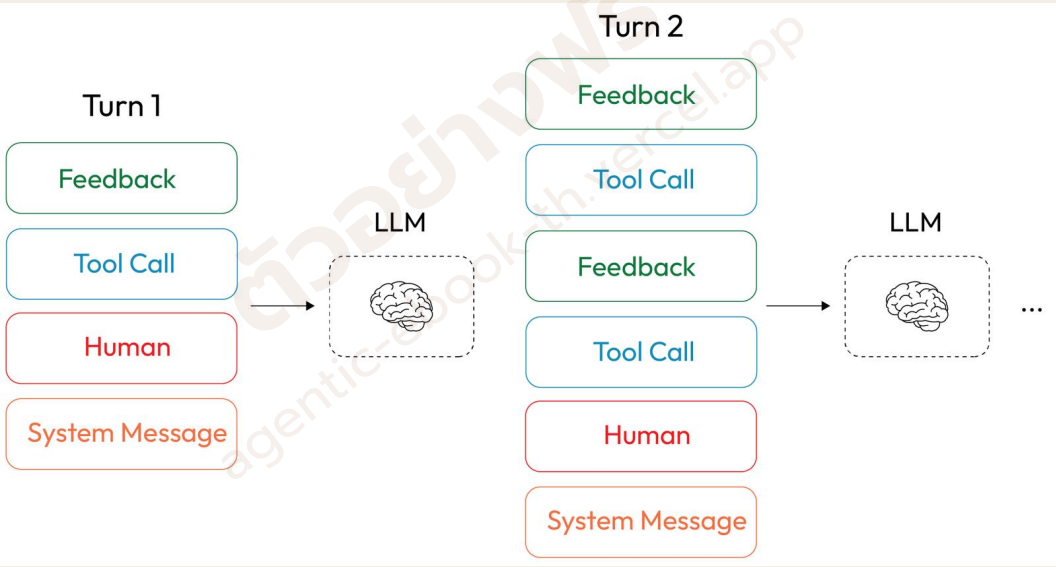
LLMs มีความสามารถใน reasoning ดีขึ้นเรื่อย ๆ เรามี tool calling และสามารถสร้าง AI agent ที่เรียกใช้ tool รับ output และวน loop จนกว่างานจะเสร็จสมบูรณ์



รูปที่ 1.2 – tool-calling loop ของ LLM (ดัดแปลงจาก concept ที่ LangChain กล่าวถึง, www.langchain.com)

อย่างไรก็ตาม เมื่อเป็นงานซับซ้อนและทำงานยาวนาน เรามักสะสม feedback จาก tool call

สิ่งนี้ทำให้ context window ขยายขึ้นอย่างต่อเนื่อง และเต็มไปด้วย output จาก tool call กับ output ระหว่างทาง



รูปที่ 1.3 – การเติบโตของ context ในหลาย turn (ดัดแปลงจาก concept ที่ LangChain กล่าวถึง, www.langchain.com)

สิ่งนี้นำไปสู่ปัญหาหลายประการ:

- context window อาจเกินขีดจำกัดขนาด
- ค่าใช้จ่ายและ latency เพิ่มขึ้น
- performance ของ agent เริ่มลดลง

หากไม่ดำเนินการใด ๆ ความเสื่อมถอยนี้จะหลีกเลี่ยงไม่ได้ การอภิปรายล่าสุดได้สำรวจว่าเหตุใด context ยาวจึงล้มเหลวในทางปฏิบัติ และความเสื่อมถอยค่อย ๆ ปรากฏขึ้นอย่างไรเมื่อข้อมูลที่ไม่เกี่ยวข้องหรือขัดแย้งกันสะสมเพิ่มขึ้น หากต้องการอ่านการอภิปรายเชิงลึก โปรดดู blog post ยอดเยี่ยมชื่อ *“เหตุใด context ยาว จึง ล้มเหลว”* โดย Dan Breunig

(https://www.dbreunig.com/2025/06/22/how-contexts-fail-and-how-to-fix-them.html) หากปล่อยให้ context เติบโตโดยปราศจากโครงสร้าง การคัดเลือก หรือการควบคุม ปัญหาหลายอย่างจะเริ่มปรากฏในระบบ agentic เช่นรายการต่อไปนี้:

- **context poisoning**: เมื่อ hallucination จาก tool call หรือ LLM call เข้าสู่ context และเริ่มส่งผลต่อ output ในอนาคต
- **context confusion**: เมื่อ context ที่ไม่จำเป็นมีอิทธิพลต่อคำตอบ แม้จะไม่เกี่ยวข้องกับงาน
- **context clash**: เมื่อส่วนต่าง ๆ ของ context ขัดแย้งกัน

ในส่วนถัดไป เราจะอภิปรายเทคนิคสำหรับทำ Context Engineering ให้ดีขึ้น เทคนิคบางส่วนนำไปใช้โดย application developer เช่น ใน tool อย่าง Claude Code ส่วนเทคนิคอื่นอยู่ฝั่ง user

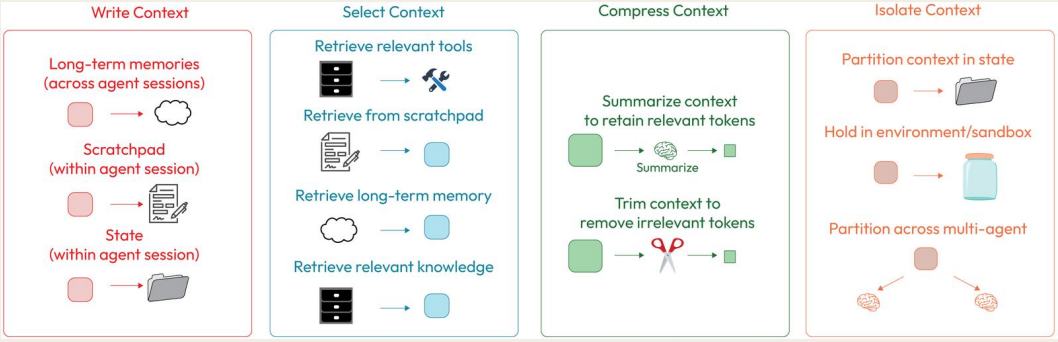
ในฐานะ user Claude Code เรามีอิทธิพลอย่างมากต่อ context ที่จะถูกส่งไปยัง LLM ในท้ายที่สุด รวมถึงคำตอบที่เราได้รับ ซึ่งหมายความว่าแม้แต่ผู้ที่ไม่ใช่ developer ก็ต้องเข้าใจ Context Engineering หากต้องการคำตอบที่ดีขึ้นจากระบบ AI และ AI agent

coding agent อย่าง Claude Code เป็นตัวอย่างที่ยอดเยี่ยมว่าเทคนิค Context Engineering ถูกนำไปใช้อย่างไรทั้งฝั่ง developer และฝั่ง user นี่คือสิ่งที่เราจะสำรวจในส่วนถัดไป รวมถึงวิธีออกแบบ context ของเราให้ดียิ่งขึ้น

Claude Code และกลยุทธ์ด้าน Context Engineering

ในส่วนนี้ เราจะกล่าวถึง Claude Code ปรัชญาด้าน Context Engineering และวิธีที่ Claude Code รับมือความท้าทายเหล่านี้ Claude Code นำกลยุทธ์ด้าน Context Engineering มาใช้ด้วยการนำไปใช้ทั้งสี่แนวทางต่อไปนี้:

- **การเขียน context**: ระบบ memory แบบคงอยู่
- **การเลือก context**: context retrieval และการแทรกอย่างชาญฉลาด
- **Context Compression**: การนำเสนอและสรุป context อย่างมีประสิทธิภาพ
- **การแยก context**: การจัดการ context แบบ multi-agent และแบบกำหนด scope



รูปที่ 1.4 – หมวดหมู่ของ Context Engineering (ดัดแปลงจาก concept ที่ LangChain กล่าวถึง, www.langchain.com)

มาเริ่มจากแนวทางแรกกัน

กลยุทธ์ที่ 1: การเขียน context และ memory แบบคงอยู่

กลยุทธ์แรกคือ **การเขียน context และ architecture memory แบบคงอยู่** Claude มีระบบ memory หลาย layer และ Claude Code นำไปใช้ลำดับชั้น memory สามระดับที่คง context ข้าม session ระหว่างเขียน code ด้วย Claude Code

- อันดับแรก เรามี **memory ของ project** (`./CLAUDE.md`) นี่คือ context ที่แชร์กันในทีมสำหรับ architecture มาตรฐาน หรือสิ่งใดก็ตามที่เกี่ยวข้องกับ project หนึ่ง ๆ memory ประเภทนี้ควบคุมด้วย version และสมาชิกทุกคนในทีมเข้าถึงได้

```
# context ของ project

## ภาพรวม architecture
นี่คือ application microservice ที่ใช้:
- Node.js พร้อม Express สำหรับบริการ API
- React พร้อม TypeScript สำหรับ frontend
- PostgreSQL พร้อม Prisma ORM
- Redis สำหรับ caching

## มาตรฐานการเขียน code
- ใช้ functional component พร้อม hook
- นำไปใช้ error boundary สำหรับ route component ทั้งหมด
- ปฏิบัติตามข้อตกลง RESTful API
- เขียน unit test สำหรับ business logic ทั้งหมด
```

- จากนั้นเรามี **memory ของ user** (`~/.claude/CLAUDE.md`) ซึ่งจัดเก็บไว้ใน home directory ของ user ภายใน directory `claude` ใน file `CLAUDE.md` โดยมีค่ากำหนดและ shortcut ส่วนบุคคลสำหรับทุก project ของ user นั้น ข้อมูลนี้จะไม่ถูก commit ไปยัง GitHub และเป็นข้อมูลเฉพาะ user แต่ละคนจะมีค่าต่างกัน และข้อมูลจะคงอยู่ในทุก Claude Code session

```
# ค่ากำหนดส่วนบุคคลสำหรับ development

## code style
- ใช้ return type ที่ชัดเจนใน TypeScript เสมอ
- เลือกใช้ const assertion แทน type annotation
- ใช้ชื่อตัวแปรที่สื่อความหมาย และหลีกเลี่ยงตัวย่อ

## shortcut workflow
- เมื่อเขียน test ให้ใช้ Jest ร่วมกับ React Testing Library
- เรียกใช้ `npm run lint` ก่อน commit เสมอ
- เลือกใช้ composition แทน inheritance
```

- สุดท้าย เรามี **การนำเข้า memory แบบ dynamic** ซึ่งช่วยให้นำเข้าจาก memory file อื่นด้วยสัญลักษณ์และ syntax `@` วิธีนี้คล้ายกับการโหลด context ทั่วไปเข้า Claude Code แต่ครั้งนี้เราสามารถมี memory file เฉพาะที่เก็บข้อมูลเจาะจงและอ้างอิงจากภายใน memory file ของเราได้

```
# ใน CLAUDE.md file ใด ๆ
@path/to/memory/file.md
@./relative/path/context.md
@~/global/user/context.md
```

เรายังปรับแต่งพฤติกรรมได้ด้วยการเขียน script ที่อัปเดต context แบบ dynamic ตาม branch Git จากนั้นเชื่อม script นี้เข้ากับ hook สำหรับสลับ context และสร้าง workflow ที่ปรับตัวได้มากขึ้น

รายละเอียดสำคัญข้อหนึ่งที่ควรจำคือ เราควรหลีกเลี่ยงการผนวกการอ้างอิง context เดิมซ้ำ ๆ ลงใน `CLAUDE.md` หาก script ใช้เพียง `>>` ในทุกการเรียกใช้ file จะขยายไม่สิ้นสุดด้วยรายการซ้ำ เพื่อป้องกันปัญหานี้ เราจะเพิ่มกลไกป้องกันขนาดเล็กที่ตรวจสอบว่ามีการอ้างอิง context อยู่แล้วหรือไม่ก่อนผนวก

ด้านล่างคือ script version ที่ปรับปรุงแล้ว ซึ่งหลีกเลี่ยงการนำเข้าซ้ำ:

```
# script สำหรับอัปเดต context แบบ dynamic ตาม branch git
#!/bin/bash
# context-switcher.sh
# โหลด context ที่เกี่ยวข้องแบบ dynamic ตาม query ของ user
# ปลอดภัยจากการนำเข้าซ้ำ

CLAUDE_MD="CLAUDE.md"

# สร้าง CLAUDE.md หากยังไม่มี
touch "$CLAUDE_MD"

add_context() {
  local context_ref="$1"
  grep -qxF "$context_ref" "$CLAUDE_MD" || echo "$context_ref" >> "$CLAUDE_MD"
}

# --- context ตาม branch ---
branch=$(git branch --show-current 2>/dev/null)

case $branch in
  "feature/auth-"*)
    add_context "@./context/auth-system.md"
    ;;
  "feature/payment-"*)
    add_context "@./context/payment-flow.md"
    ;;
  "hotfix/"*)
    add_context "@./context/production-hotfix.md"
    ;;
esac

# --- context ตาม query ---
user_input="$1"

if [[ -n "$user_input" ]]; then
  if [[ $user_input == *"database"* || $user_input == *"migration"* ]]; then
    add_context "@./context/database-context.md"
  elif [[ $user_input == *"API"* || $user_input == *"endpoint"* ]]; then
    add_context "@./context/api-context.md"
  elif [[ $user_input == *"frontend"* || $user_input == *"component"* ]]; then
    add_context "@./context/frontend-context.md"
  fi
fi
```

การเปลี่ยนแปลงสำคัญคือ function helper `add_context` ซึ่งใช้ `grep -qxF` เพื่อตรวจสอบว่ามีการอ้างอิง context อยู่ใน `CLAUDE.md` แล้วหรือไม่ หากมีอยู่แล้ว จะไม่มีการผนวกสิ่งใด วิธีนี้ทำให้ script เป็น idempotent และปลอดภัยต่อการเรียกใช้ทุกครั้งที่ส่ง prompt

จากนั้นเราเชื่อม script นี้เข้ากับ hook

>_ agentic-ebook-th.vercel.app

จบช่วงตัวอย่างของบทที่ 1

บทที่ 1 มีทั้งหมด 12 หน้า — คุณเพิ่งอ่าน 6 หน้าแรก

อ่านต่อทั้งบทได้ในฉบับเต็ม

ตัวอย่างถัดไป — บทที่ 2

พลิกหน้าถัดไปเพื่อดูต่อ

2

สาระสำคัญของ Claude Code

บทนี้จะช่วยให้คุณคุ้นเคยกับ Claude Code ด้วยการพาไปดูวิธีทำงานจริงในทางปฏิบัติ แทนที่จะกระโดดเข้าสู่ feature ขั้นสูงทันที เราจะเริ่มจากสร้างความเข้าใจในตัวระบบ คำศัพท์ที่ระบบใช้ และวิธีที่ระบบโต้ตอบกับ codebase ระหว่างทาง คุณจะเห็นว่า Claude Code สังเกต project สร้าง context และใช้ context นั้นช่วยทำ task เขียน code อย่างไร

เราเริ่มด้วยการตั้งค่า Next.js project แบบง่ายและทำ integration เข้ากับ Claude Code ซึ่งเป็นวิธีเชิงปฏิบัติสำหรับสำรวจ concept ต่าง ๆ เช่น file `CLAUDE.md` ระบบสิทธิ์ และเหตุใดการจัดการ context อย่างรอบคอบจึงสำคัญ นอกจากนี้ เรายังดูวิธีขยาย Claude Code ด้วย tool ภายนอกผ่าน **Model Context Protocol (MCP)** และความสามารถที่ MCP เพิ่มให้ workflow ประจำวัน

บทนี้ยังแนะนำ spec-driven design ในฐานะวิธีกำหนดพฤติกรรมของ Claude ก่อนจะเขียน code ใด ๆ การสร้างและอ้างอิง spec ช่วยให้เราส่งผลต่อคุณภาพและความสม่ำเสมอของ output ที่ได้รับกลับมา ใน process นี้ เราจะเริ่มสร้าง project ชื่อ HookHub project ที่คุณสร้างในบทนี้จะอยู่กับเราไปตลอดทั้งเล่ม และเป็นจุดอ้างอิงร่วมกันเมื่อเข้าสู่ agentic workflow ที่ซับซ้อนยิ่งขึ้นในภายหลัง

บทนี้จะครอบคลุมหัวข้อต่อไปนี้:

- การตั้งค่า Next.js project
- การเชื่อมต่อ Playwright MCP กับ Claude Code
- การใช้ spec-driven design
- การทำ implementation page หลักด้วย spec-driven design

เอาละ มาเริ่มกันเลย!

การติดตั้ง Claude Code

ก่อนดำเนินการต่อ ตรวจสอบให้แน่ใจว่าได้ติดตั้ง Claude Code บนเครื่องของคุณแล้ว

ขณะนี้ Claude Code มี installer native ที่ตรวจจับระบบปฏิบัติการและติดตั้ง version ที่เหมาะสมโดยอัตโนมัติ โปรดทำตามคู่มือการติดตั้งอย่างเป็นทางการ:

https://code.claude.com/docs/en/setup#native-install-recommended

installer จะจัดการตั้งค่าและอัปเดตตามประเภทเครื่องของคุณ เมื่อติดตั้งเสร็จ สามารถตรวจสอบได้ด้วยการเรียกใช้:

```
claude --version
```

หลังยืนยันว่าติดตั้ง Claude Code ถูกต้องแล้ว ให้ดำเนินการตั้งค่า Next.js ด้านล่างต่อ

คำสั่ง terminal ทั้งหมดในบทนี้สาธิตบน macOS สำหรับ Windows รุ่นปัจจุบัน Claude Code สามารถใช้ PowerShell เป็น shell หลักได้โดยไม่ต้องติดตั้ง Git Bash ส่วน command ที่เป็น Unix-specific อาจต้องปรับเป็น PowerShell syntax หรือเลือกใช้ WSL ตามความเหมาะสม `npx`, `npm` และ `claude` ยังคงใช้ได้เมื่อ runtime ที่เกี่ยวข้องอยู่ใน `PATH`

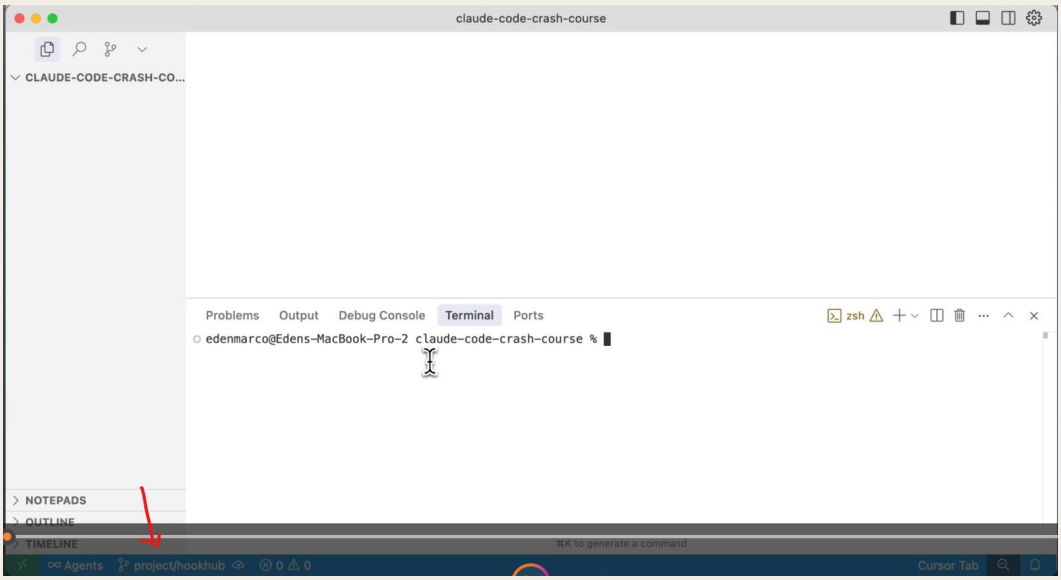
การตั้งค่า Next.js project

ในส่วนนี้ เราจะตั้งค่า Next.js project ด้วยตนเอง แม้จะส่ง prompt ให้ Claude สร้าง Next.js project ได้ แต่ผมเลือกทำด้วยตนเองเนื่องจากปัญหาสิทธิ์ในเครื่องและ configuration ที่ต้องจัดการล่วงหน้า สำหรับผู้ที่ต้องการ Claude สามารถเรียกใช้คำสั่ง Bash เหล่านี้ได้อย่างเต็มที่เมื่อได้รับ syntax ที่ถูกต้อง

เราเริ่มด้วยการตั้งค่า Next.js project คุณสามารถทำตามคู่มือการติดตั้ง Next.js อย่างเป็นทางการ (https://nextjs.org/docs/app/getting-started/installation) หรือวางคำสั่งต่อไปนี้:

```
npx create-next-app@latest
```

ดังที่เห็นจากภาพหน้าจอต่อไปนี้ เราอยู่ใน branch **project/HookHub**



รูปที่ 2.1 – branch project/hookhub

เราวางคำสั่งและ scaffold Next.js project ระบบจะแสดงตัวเลือกให้ดำเนินการต่อดังนี้:

```
edenmarco@Edens-MacBook-Pro-2 claude-code-crash-course % npx create-next-app@latest
Need to install the following packages: (จำเป็นต้องติดตั้ง package ต่อไปนี้:)
create-next-app@15.4.6
Ok to proceed? (y) (ตกลงดำเนินการต่อหรือไม่)
```

หลังวางคำสั่ง คุณอาจพบ error ซึ่งเกิดจาก user ปัจจุบันไม่มีสิทธิ์ที่จำเป็นสำหรับติดตั้ง package แบบ global บนระบบ

มีวิธีแก้ไขสองวิธี วิธีหนึ่งคือเรียกใช้คำสั่งด้วย `sudo` อีกวิธีคือปรับสิทธิ์ของ `npm` เพื่อให้ติดตั้งได้โดยไม่ต้องยกระดับสิทธิ์ เพื่อความเรียบง่าย เราจะเรียกใช้คำสั่งอีกครั้งด้วย `sudo` ซึ่งมอบสิทธิ์ที่จำเป็น

เมื่อดำเนินการด้วยสิทธิ์ที่ถูกต้อง การ scaffold Next.js project จะดำเนินการสำเร็จ ระบบจะขอให้คุณยืนยันตัวเลือก configuration ให้เลือกตัวเลือกเริ่มต้นแล้วดำเนินการต่อ จากนั้นระบบจะติดตั้ง dependency และสร้าง code boilerplate สำหรับ Next.js project

เมื่อติดตั้งเสร็จ ให้เข้าไปใน project directory ที่เพิ่งสร้าง:

```
cd hookhub
```

ตอนนี้ให้เริ่ม development server:

```
npm run dev
```

หากพยายามเรียกใช้ server จาก directory ที่ไม่ถูกต้อง คำสั่งจะล้มเหลว ตรวจสอบให้แน่ใจว่าคุณอยู่ภายใน project directory ก่อนเรียกใช้

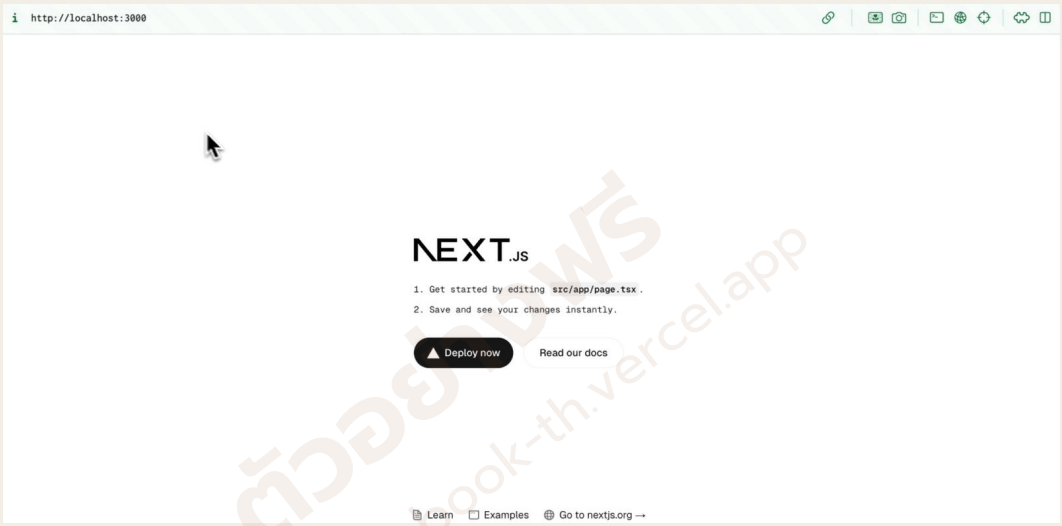
เมื่อ server เริ่มทำงานสำเร็จ ให้เปิด `http://localhost:3000` ใน browser คุณควรเห็น application boilerplate เริ่มต้นของ Next.js

ณ จุดนี้ Next.js project ทำงานอยู่บนเครื่อง local และพร้อมทำ integration กับ Claude Code หลังเรียกใช้คำสั่งอีกครั้ง เราจะสร้าง project hookhub เลือก **Yes** สำหรับ prompt ต่าง ๆ และติดตั้ง dependency ที่จำเป็นทั้งหมด เมื่อตั้งค่าเสร็จ คุณจะเห็น code boilerplate ของ Next.js project

การเรียกใช้และตรวจสอบ application

เมื่อตั้งค่าทุกอย่างแล้ว เราจะเข้าไปใน directory hookhub และเรียกใช้ project ด้วย `npm run dev`

ตอนนี้ Next.js application กำลังทำงาน ให้เปิด `localhost:3000` แล้วคุณจะเห็น application boilerplate มาตรฐานของ Next.js ดังภาพ:

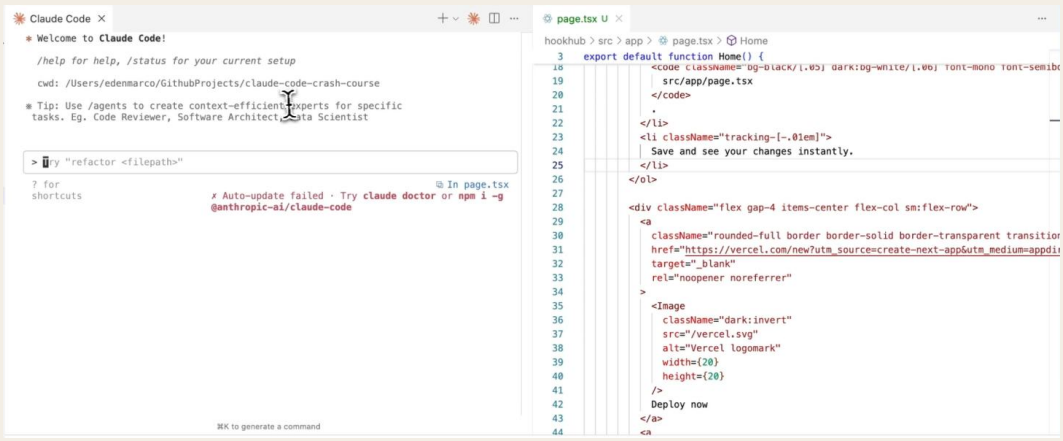


รูปที่ 2.2 – Next.js application เริ่มต้นทำงานบน localhost:3000

การเริ่มต้น Claude Code

ก่อนทำการเปลี่ยนแปลงเพิ่มเติม ให้กลับไปที่ Claude Code และมอบ context ของ project ให้ระบบ โดยเริ่มต้น file `CLAUDE.md` เพื่อเพิ่มเข้าไปใน context ของ project

Claude Code เปิดผ่าน integration ของ Cursor โดยส่วนตัวแล้ว ผมชอบวาง Claude Code ไว้ตรงกลางและวางโปรแกรมแก้ไข code ไว้ด้านขวาดังที่แสดง การจัดวางนี้ทำให้ Claude Code ทำหน้าที่เป็นตัวขับเคลื่อนหลัก

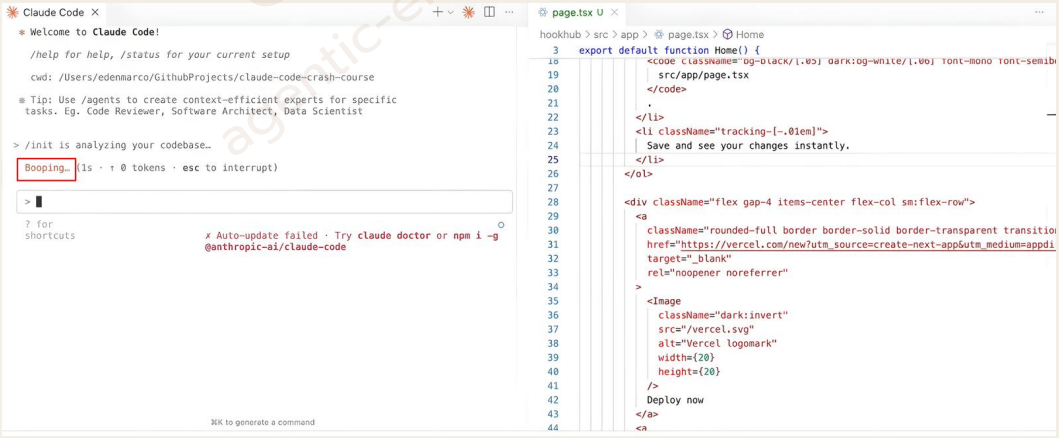


รูปที่ 2.3 – Claude Code เปิดผ่าน integration ของ Cursor

ขั้นตอนแรกเมื่อเริ่มต้น project ด้วย Claude Code คือเรียกใช้คำสั่ง `/init` ไม่ว่า project จะใหม่หรือมี file อยู่แล้วจำนวนมากก็ใช้วิธีเดียวกัน เมื่อดำเนินการ `/init` Claude จะสแกน file ตรวจสอบ content และเริ่มสร้างข้อสังเกตเกี่ยวกับ codebase

Claude จะระบุ tech stack content หลัก และโครงสร้าง directory ข้อมูลทั้งหมดนี้จะถูกบันทึกไว้ใน file `CLAUDE.md` ซึ่งเราจะสำรวจในช่วงหลังของหนังสือ ในที่สุด Markdown file เหล่านี้จะกระจายอยู่ทั่ว project เพื่อรองรับ Context Engineering วัตถุประสงค์ของ file เหล่านี้คือส่ง context คุณภาพสูงไปยัง LLM ในทุก request ทำให้ LLM สร้าง code ที่แม่นยำและมีประโยชน์ได้ง่ายขึ้น

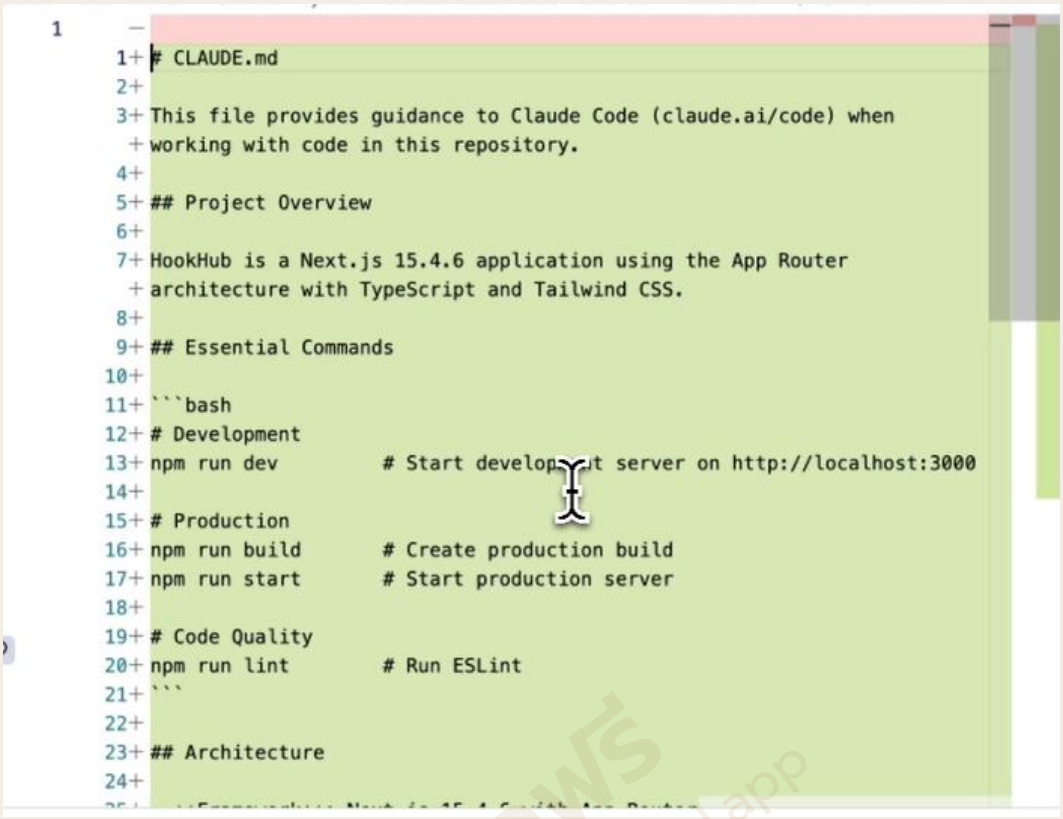
ขณะ Claude กำลังทำงาน status จะแสดงว่าอยู่ในสถานะ **Booping**



รูปที่ 2.4 – สถานะ Booping ใน integration ของ Cursor

ใน context นี้ Claude กำลังวิเคราะห์ repository สำรวจโครงสร้าง directory และอ่าน content file หลังจากนั้น ระบบจะสร้าง task เพื่อวิเคราะห์โครงสร้าง codebase และดำเนินการชุดคำสั่งแบบ batch เพื่อ list และอ่าน file

โปรดทราบว่า process นี้อาจใช้เวลานานขึ้นสำหรับ project ขนาดใหญ่ เมื่อเสร็จแล้ว Claude จะเตรียมสร้าง file `CLAUDE.md` ตามข้อมูลที่รวบรวมไว้ และสร้าง output ดังต่อไปนี้:



รูปที่ 2.5 – CLAUDE.md file ที่สร้างขึ้น พร้อมภาพรวม project และคำสั่งต่าง ๆ

ดังที่เห็น มีการสร้าง file `CLAUDE.md` ซึ่งประกอบด้วยโครงสร้าง project และรายละเอียดที่เกี่ยวข้อง

เพื่อบันทึก file Claude จะขอสิทธิ์ เราสามารถอนุญาตให้เขียน file และแก้ไขต่อโดยไม่ถามอีก หรือปฏิเสธการเปลี่ยนแปลงหาก ต้องการแก้ไขหรือยกเลิก ระบบสิทธิ์นี้ใช้กับทุก file ใหม่หรือทุกการแก้ไข

```
Opened changes in Cursor (เปิดการเปลี่ยนแปลงใน Cursor)
Save file to continue... (บันทึก file เพื่อดำเนินการต่อ...)

Do you want to make this edit to CLAUDE.md? (คุณต้องการแก้ไข CLAUDE.md ตามนี้หรือไม่)
1. Yes (ใช่)
2. Yes, and don't ask again this session (shift+tab) (ใช่ และไม่ต้องถามอีกใน session นี้)
3. No, and tell Claude what to do differently (esc) (ไม่ และแจ้ง Claude ว่าควรทำอย่างไรให้ต่างออกไป)
```

นี่เป็นส่วนสำคัญของการฝึก Agentic Coding อย่างปลอดภัย permission สิทธิ์แบบไม่จำกัดอาจนำไปสู่การเปลี่ยนแปลง file โดยไม่ตั้งใจและ bug ได้อย่างรวดเร็ว การรักษาการควบคุมช่วยให้ workflow ปลอดภัยยิ่งขึ้น

เลือก **Yes** ในตอนนี้ และเลือกไม่ให้ระบบถามอีกระหว่าง session นี้ Claude จะทำ process จนเสร็จและสร้าง file `CLAUDE.md`

เนื้อหาของ CLAUDE.md file

file นี้ประกอบด้วย architecture โครงสร้าง และข้อมูลพื้นฐานของ project เช่นรายการต่อไปนี้:

- เป้าหมายของ project
- ปัญหาทางธุรกิจที่กำลังแก้ไข

จบช่วงตัวอย่างของบทที่ 2

บทที่ 2 มีทั้งหมด 17 หน้า — คุณเพิ่งอ่าน 6 หน้าแรก

อ่านต่อทั้งบทได้ในฉบับเต็ม

ตัวอย่างถัดไป — บทที่ 3

พลิกหน้าถัดไปเพื่อดูต่อ

3

เริ่มต้นใช้งาน Claude Code – พาชม command สำคัญ

ในบทนี้ เราจะต่อยอดจากการเริ่มต้นใช้งาน Claude Code ไปสู่การกำหนดค่าเพื่อใช้ทำงานในแต่ละวัน เป้าหมายคือช่วยให้คุณคุ้นเคยกับพฤติกรรมของ Claude Code ในการใช้งานจริง วิธีควบคุม Claude Code ภายใน session และวิธีตั้งค่าให้ทำงานอย่างสอดคล้องกันในทุก project

เราจะกล่าวถึงราคาและการยืนยันตัวตนโดยสังเขป เพื่อให้คุณเลือกการตั้งค่าที่เหมาะสมก่อนเริ่มต้น หลังจากนั้น เราจะมุ่งเน้นส่วนที่คุณต้องใช้อยู่เสมอ ได้แก่ command สำหรับจัดการ session และ context configuration ในระดับ user และระดับ project ตลอดจนการเรียกใช้ Claude Code โดยตรงภายใน IDE อย่าง Cursor จากนั้นเราจะต่อยอดพื้นฐานเหล่านี้ด้วย feature ที่ช่วยให้ workflow ซึ่งใช้เวลานานมี security และเชื่อถือได้ยิ่งขึ้น ซึ่งรวมถึง hook สำหรับ automation memory ผ่าน file `CLAUDE.md` Rewind feature ผ่านการทำ checkpoint และ Skills (skill) สำหรับ workflow ที่ทำซ้ำได้และมี scope ชัดเจน

> คุณสามารถเรียกใช้ skill ด้วยตนเองผ่าน syntax `/skill-name` เอกสาร version เก่าอาจเรียกกลไกนี้ว่า *custom commands* แต่ Claude Code ปัจจุบันรวม workflow ที่เรียกใช้ซ้ำได้ไว้ภายใต้ **Skills** และยังรองรับ file เดิมใน `.claude/commands/` เพื่อความเข้ากันได้

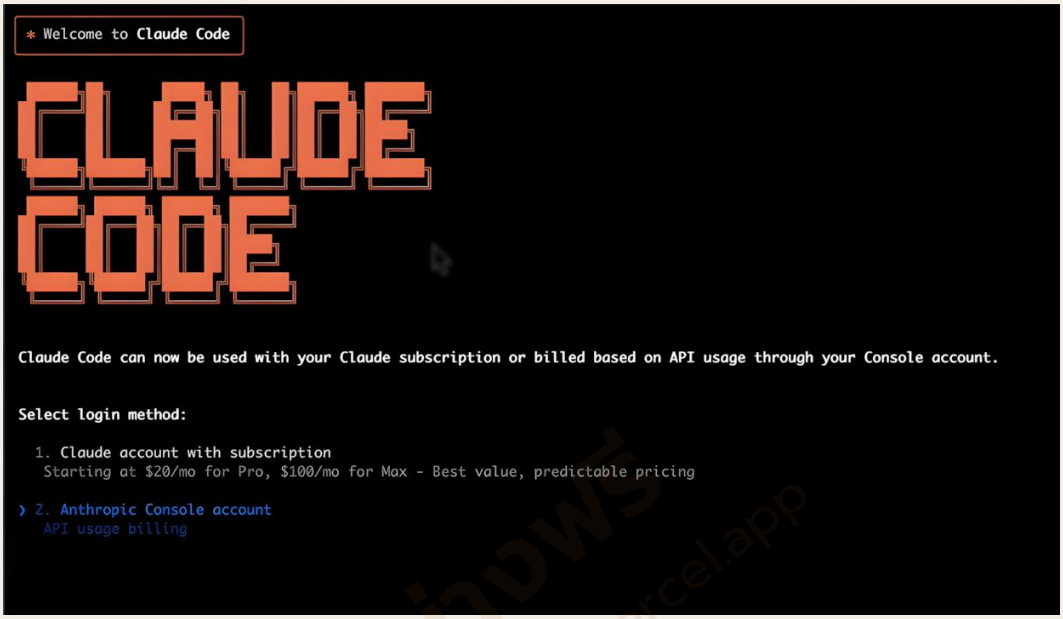
บทนี้จะครอบคลุมหัวข้อต่อไปนี้:

- ราคาและ plan ของ Claude Code
- การควบคุม Claude Code ด้วย command
- การใช้ hook ใน Claude Code
- การทำความเข้าใจ memory ใน Claude Code
- Rewind feature ใน Claude Code
- skill ใน Claude Code

ราคาและ plans ของ Claude Code

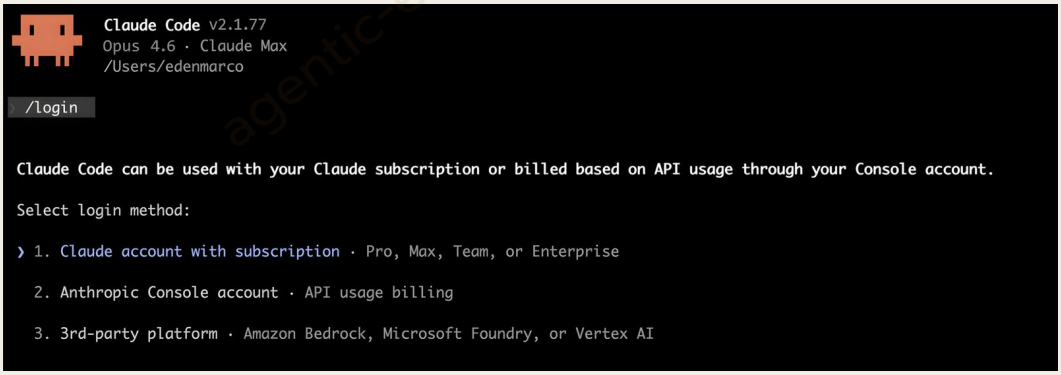
ในส่วนนี้ เราจะอธิบายตัวเลือกด้านราคาและ plan ต่าง ๆ ที่มีให้สำหรับการใช้งานและใช้บริการ Claude Code เราจะเริ่มจากการยืนยันตัวตน เนื่องจากวิธีที่คุณใช้ยืนยันตัวตนจะเป็นตัวกำหนดวิธีเรียกเก็บเงินตามการใช้ token ด้วย

> รายละเอียดราคาและ plan ที่กล่าวถึงในส่วนนี้ถูกต้อง ณ เวลาที่เขียน เนื่องจากระดับการสมัครสมาชิก ขีดจำกัดการใช้งาน และนโยบายการเรียกเก็บเงินอาจเปลี่ยนแปลงได้ โปรดตรวจสอบหน้าราคาอย่างเป็นทางการของ Claude เสมอเพื่อดูข้อมูลล่าสุด: https://claude.com/pricing



รูปที่ 3.1 – หน้าจอตัวเลือกการยืนยันตัวตนของ Claude Code

Claude Code มีวิธีหลัก 3 วิธีสำหรับยืนยันตัวตนและชำระค่า token ที่คุณใช้



รูปที่ 3.2 – วิธี login สำหรับการสมัครสมาชิก Claude Code และการเรียกเก็บเงินผ่าน API

ตัวเลือกที่ 1: การใช้ API key ของ Anthropic

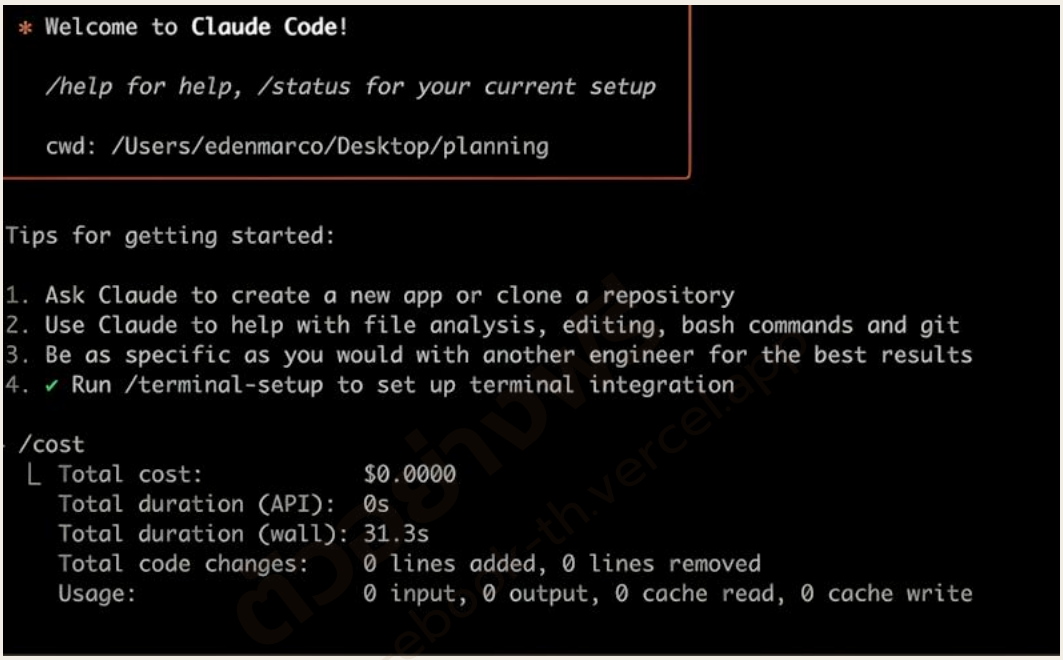
ตัวเลือกหนึ่งคือยืนยันตัวตนด้วย API key ของ Anthropic วิธีนี้ใช้ model การเรียกเก็บเงินแบบ pay-as-you-go โดยติดตามการใช้งานผ่าน console Anthropic ตัวเลือกนี้เหมาะสำหรับ user ที่ต้องการชำระเงินตามการใช้งาน หรือต้องการทดลองประเมิน Claude Code ก่อนตัดสินใจสมัครสมาชิก

เมื่อใช้วิธี API key ของ Anthropic ขณะทำตามหนังสือเล่มนี้ ปริมาณการใช้งานจะแตกต่างกันไปตามความถี่ที่เรียกใช้ Claude Code และ model ที่เลือกใช้

หากคุณเลือกตัวเลือกนี้ ให้เลือกวิธียืนยันตัวตนด้วย API key และอนุญาต Claude Code ผ่าน organization Anthropic ของคุณ จากนั้น Claude Code จะสร้าง API key และกำหนดค่าให้โดยอัตโนมัติใน terminal เมื่อดำเนินการเสร็จแล้ว Claude Code จะทำงานด้วย model pay-as-you-go

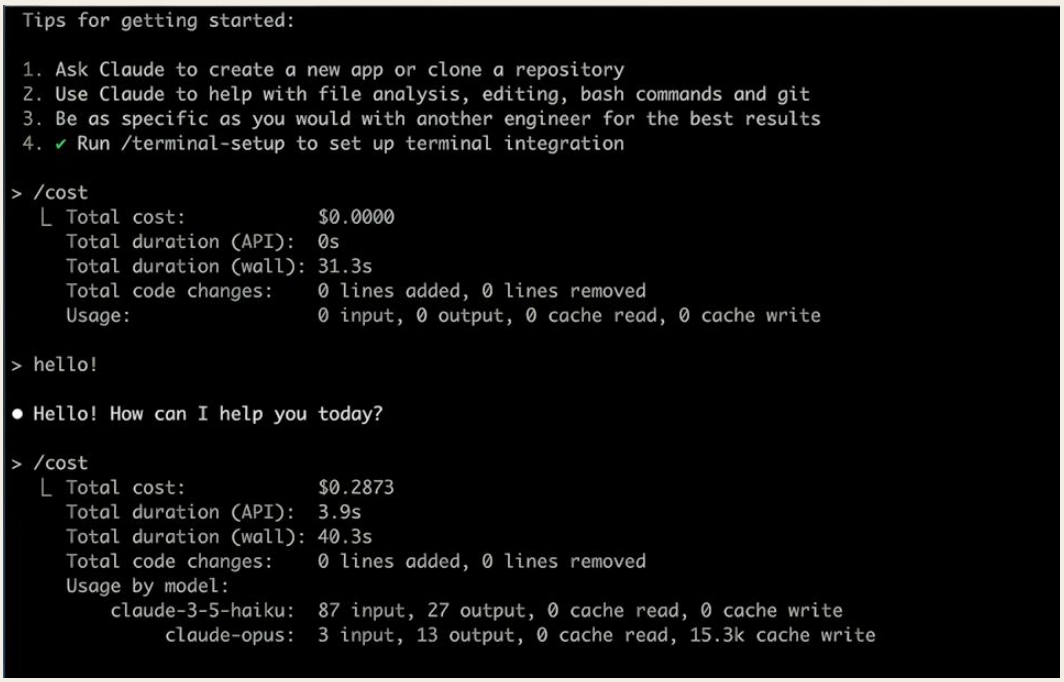
ประเด็นที่ควรคำนึงถึงเมื่อใช้ Anthropic API

เมื่อต้องการติดตามการใช้งาน ให้เรียก command `/usage` ซึ่งแสดง session cost, plan usage limit และ activity ที่เกี่ยวข้อง โดย `/cost` ยังคงใช้ได้ในฐานะ alias ของ `/usage`



รูปที่ 3.3 – การติดตามการใช้ token ด้วย command /cost

ข้อมูลการใช้ API key ดูได้ผ่าน Anthropic console เช่นกัน เพียงพิมพ์ข้อความใดก็ได้ แล้วเมื่อได้รับ output ให้พิมพ์ `/usage` อีกครั้ง



รูปที่ 3.4 – ข้อมูลการใช้ token ที่อัปเดตหลังจากเรียกใช้ prompt

โปรดจำไว้ว่าราคาจะแตกต่างกันตาม model ที่ใช้ ตัวอย่างเช่น การใช้ Opus ทำให้มีค่า token สูงกว่า model อื่น

ข้อควรพิจารณาที่สำคัญเมื่อใช้ API key

การใช้ API key จำเป็นต้องทำด้วยความระมัดระวัง เพราะมีความเสี่ยงที่ key อาจถูกเปิดเผยโดยไม่ตั้งใจ ซึ่งจะทำให้ผู้อื่นนำไปใช้ token โดยคุณเป็นผู้รับภาระค่าใช้จ่าย ในกรณีร้ายแรง อาจทำให้เกิดค่าใช้จ่ายสูงเกินคาด

นอกจากนี้ หนังสือเล่มนี้ยังครอบคลุมการทำงานกับ agent subagent และ pattern orchestration ซึ่งทั้งหมดสามารถใช้ token เป็นจำนวนมาก หากมีส่วนใดกำหนดค่าผิดพลาด ก็อาจเรียก agent จำนวนมากพร้อมกันและทำให้ค่าใช้จ่ายเพิ่มขึ้นอย่างมาก

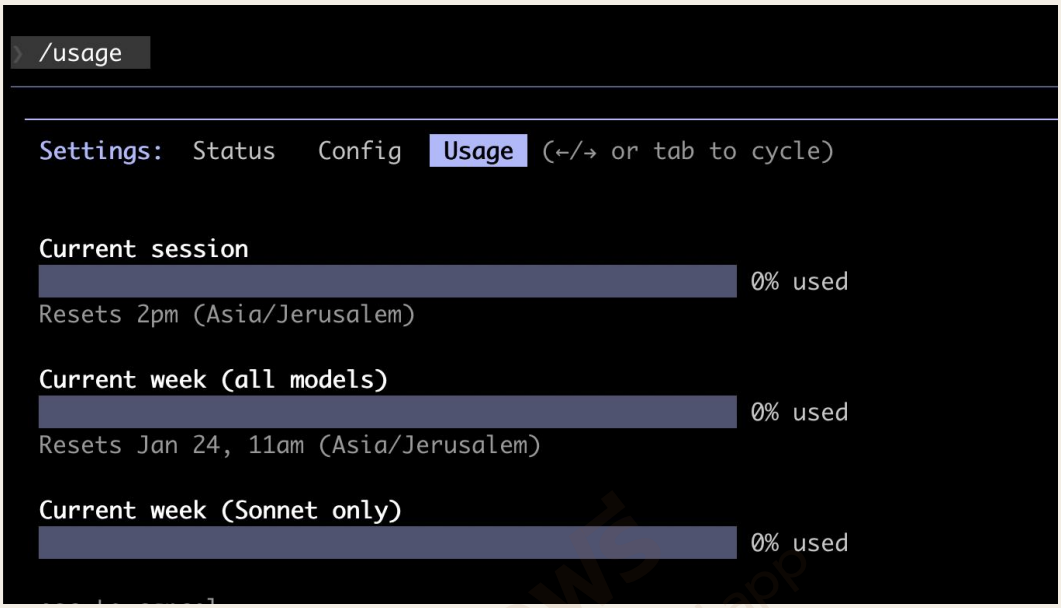
วิธีหนึ่งในการลดความเสี่ยงนี้คือกำหนดขีดจำกัดค่าใช้จ่ายใน Anthropic console โดย Anthropic อนุญาตให้คุณกำหนดขีดจำกัดการใช้งานสำหรับ account ได้ และเมื่อถึงขีดจำกัดแล้ว ระบบจะไม่ให้บริการ request เพิ่มเติม สำหรับรายละเอียดเกี่ยวกับการกำหนดค่าขีดจำกัดเหล่านี้ โปรดดูหน้า Settings ของ Anthropic (https://platform.claude.com/settings/limits)

ตัวเลือกที่ 2: การใช้การสมัครสมาชิก Claude

หากคุณทำตามการตั้งค่าใน *บทที่ 2* คุณกำลังใช้ตัวเลือกที่ 2 ซึ่งยืนยันตัวตนผ่านการสมัครสมาชิก Claude เราจะใช้ตัวเลือกนี้ตลอดทั้งเล่ม กล่าวคือการยืนยันตัวตนผ่านการสมัครสมาชิก Claude ด้วยแนวทางนี้ คุณจะเชื่อมต่อการสมัครสมาชิก Claude ที่มีอยู่กับ Claude Code โดยตรง ทำให้ใช้งานร่วมกันได้อย่างราบรื่น พร้อมค่าใช้จ่ายรายเดือนที่คาดการณ์ได้

Claude มี plan หลายระดับ และสิทธิ์ของ Claude Code แตกต่างกันตาม plan, organization policy, provider และภูมิภาค เนื่องจากราคา quota และ feature availability เปลี่ยนแปลงได้รวดเร็ว ให้ตรวจสอบหน้า plan อย่างเป็นทางการและใช้ `/usage` เพื่อดู limit ของ account ที่กำลังใช้งาน แทนการยึดตัวเลขราคาในหนังสือเป็นข้อมูลถาวร

ทีนี้ สิ่งที่ต่างจากตัวเลือกก่อนหน้าคือ สำหรับ subscription plan การติดตามการใช้งานจะทำงานต่างออกไป เนื่องจาก Claude Code รวมอยู่ในการสมัครสมาชิกแล้ว `/usage` จึงเน้น plan limit และ activity มากกว่าค่าใช้จ่ายระดับ token แม้ว่ายังมีขีดจำกัดการใช้งานตาม plan อยู่ก็ตาม



รูปที่ 3.5 – การดูปริมาณการใช้การสมัครสมาชิกด้วย command /usage

model ที่ใช้ได้ภายใต้ตัวเลือกนี้

การใช้ command `/model` จะแสดง model ที่ account, organization policy และ provider ปัจจุบันอนุญาต Claude Code รองรับ model หลายตระกูล รวมถึง Opus, Sonnet และ Haiku โดย default model และ context window อาจต่างกันตาม plan และ provider

- **Claude Opus**: เหมาะกับ task ที่ต้องใช้ reasoning ลึก การตัดสินใจเชิง architecture และ effort สูง
- **Claude Sonnet**: สร้างสมดุลระหว่าง capability, speed และ usage เหมาะกับ development workflow ประจำวัน ณ วันที่ 10 กรกฎาคม 2026 Sonnet 5 มี native context window ขนาด 1M token และ adaptive thinking ใน environment ที่รองรับ
- **Claude Haiku**: เหมาะกับ task ที่เรียบง่าย latency ต่ำ และ worker ที่ต้องประหยัด usage

ให้เลือก model ตามความซับซ้อน latency, provider availability และ verification requirement แทนการผูก workflow ทั้งหมดกับ model เดียว รายละเอียด model รุ่นปัจจุบัน, alias และ fallback chain อยู่ในบทที่ 12

ตัว model ได้รับการ host และให้บริการโดย Anthropic สำหรับ environment ระดับองค์กร ยังสามารถเข้าถึง model ผ่าน third-party provider ที่รองรับ เช่น Amazon Bedrock, Google Cloud's Agent Platform หรือ Microsoft Foundry ได้ด้วย capability และ model ที่ใช้ได้อาจแตกต่างตาม provider การตั้งค่าลักษณะนี้มักใช้ในองค์กรที่ต้องการระบบเรียกเก็บเงินแบบรวมศูนย์ control ด้าน compliance เฉพาะ cloud ตัวเลือกการนำขึ้นใช้งานตามภูมิภาค หรือ governance ผ่าน cloud infrastructure ที่มีอยู่

subscription plan และราคา

Claude มีการสมัครสมาชิกหลายระดับ

- plan Free
- plan Pro
- plan Max 5x
- plan Max 20x

มีการนำ rate limit มาใช้เพื่อป้องกันการใช้งานในทางที่ผิด เช่น การแชร์ข้อมูลยืนยันตัวตนหรือการใช้งานระบบอย่างไม่เหมาะสม ขีดจำกัดเหล่านี้มีผลกับทั้ง plan Pro และ Max

สำหรับ user ส่วนใหญ่ โดยเฉพาะผู้ที่ทำเป็นงานอดิเรก แนะนำให้เริ่มจาก plan Pro การอัปเกรดเป็น plan Max เหมาะสมก็ต่อเมื่อต้องการปริมาณการใช้งานที่สูงขึ้น

ขีดจำกัดการใช้งานที่แน่นอนอาจระบุเป็นตัวเลขได้ยาก และเรื่องนี้ไม่ได้เกิดขึ้นเฉพาะกับ Claude Code การกำหนดราคาสำหรับ coding agent AI ยังเป็นความท้าทายต่อเนื่องทั่วทั้งอุตสาหกรรม การเปลี่ยนแปลง model ราคาได้สร้างความไม่พอใจใน community developer และเรื่องนี้ยังคงเป็นด้านที่พัฒนาเปลี่ยนแปลงอยู่

ส่วนนี้มีจุดประสงค์เพื่อชี้แจงตัวเลือกด้านราคาที่มีอยู่ และช่วยให้คุณเข้าใจวิธีเรียกเก็บเงินเมื่อใช้ Claude Code เพื่อให้เลือกแนวทางที่เหมาะกับความต้องการของคุณมากที่สุด

ตัวเลือกที่ 3: การใช้ผู้ให้บริการ cloud บุคคลที่สาม

Claude Code ยังสามารถยืนยันตัวตนผ่าน third-party provider ที่รองรับ เช่น Amazon Bedrock, Google Cloud's Agent Platform หรือ Microsoft Foundry ในกรณีนี้ การเรียกเก็บเงินและการใช้งานจะได้รับการจัดการผ่าน provider ที่เลือก แทนที่จะผ่านการสมัครสมาชิก Claude หรือ API key ของ Anthropic โดยตรง สำหรับรายละเอียด integration และ capability ที่รองรับ โปรดดูเอกสารอย่างเป็นทางการ: https://code.claude.com/docs/en/third-party-integrations

จบช่วงตัวอย่างของบทที่ 3

บทที่ 3 มีทั้งหมด 30 หน้า — คุณเพิ่งอ่าน 6 หน้าแรก

อ่านต่อทั้งบทได้ในฉบับเต็ม

ตัวอย่างถัดไป — บทที่ 12 (อัปเดต 2026)

พลิกหน้าถัดไปเพื่อดูต่อ

12

Claude Code รุ่นปัจจุบัน: จาก autonomous workflow สู่ production-ready operations

ในบทก่อนหน้า เราได้สร้างรากฐานตั้งแต่ Context Engineering, MCP, Hooks, Skills, subagents, Git worktrees ไปจนถึง deep agent แล้ว บทนี้จึงไม่ย้อนกลับไปอธิบาย concept เหล่านั้นใหม่ แต่จะพิจารณาเฉพาะสิ่งที่เปลี่ยนวิธีทำงานอย่างมีนัยสำคัญใน Claude Code รุ่นปัจจุบัน โดยเรียบเรียงจากเอกสาร **What's New** ของ Anthropic ตั้งแต่ Week 13 ถึง Week 28 ปี 2026

การเปลี่ยนแปลงรอบนี้สะท้อนทิศทางที่ชัดเจน Claude Code ไม่ได้พัฒนาเพียงด้าน model หรือคุณภาพการเขียน code แต่กำลังกลายเป็น development runtime ที่จัดการ session หลายรายการ ทำงานแบบ background ตรวจสอบผลลัพธ์ผ่าน browser และ native GUI รักษา safety boundary ระหว่าง autonomous execution และ production infrastructure ตลอดจนเผยแพร่ output เป็น Artifact ที่ผู้ร่วมงานเปิดดูได้

บทนี้ใช้ข้อมูลที่เผยแพร่ถึง **10 กรกฎาคม 2026** ครอบคลุม Claude Code v2.1.83 ถึง v2.1.206 ตาม weekly digest อย่างเป็นทางการ เนื่องจาก Claude Code มี release cadence ที่รวดเร็ว คุณควรตรวจสอบ Claude Code What's New และ Commands reference อีกครั้งก่อนนำ configuration ที่เกี่ยวข้องกับ plan, model, provider หรือ preview feature ไปใช้ใน production

บทนี้จะครอบคลุมหัวข้อต่อไปนี้:

- model และ Context Engineering รุ่นปัจจุบัน
- orchestration ของ session, subagent และ workflow ระยะยาว
- verification loop ผ่าน browser, native GUI, Artifact และ code review
- safe autonomy ด้วย Auto mode, permission rules และ security review
- operation, configuration และ troubleshooting สำหรับ environment ที่ขยายตัว
- workflow ตัวอย่างสำหรับยกระดับ HookHub จาก task เดี่ยวเป็น production-ready autonomous workflow

สิ่งที่เปลี่ยนไปและวิธีอ่านบทนี้

ถ้ามองเฉพาะรายการ feature การอัปเดตระหว่าง Week 13 ถึง Week 28 อาจดูเหมือนชุด command จำนวนมาก แต่เมื่อจัดกลุ่มตามหน้าที่ จะเห็นการเปลี่ยนแปลงของ architecture 4 ชั้น:

1. **reasoning layer** เปลี่ยนด้วย Claude Sonnet 5, context window 1M token, adaptive thinking และ model fallback chain
2. **orchestration layer** เพิ่ม Agent view, background subagent, nested subagent, `/goal`, Dynamic workflows, Monitor และ Routines
3. **interaction layer** เพิ่ม Claude in Chrome, Desktop in-app Browser, Computer use และ Artifacts
4. **governance layer** เพิ่ม Auto mode, parameter-aware permission rules, protected paths, `security-guidance`, `/doctor` และ Safe mode

การเปลี่ยนแปลงเหล่านี้มีผลร่วมกัน เมื่อ context ใหญ่ขึ้น agent สามารถทำงานนานขึ้น แต่ไม่ได้หมายความว่าควรใส่ทุกอย่างไว้ใน conversation เดียว เมื่อ orchestration ทำงานแบบ background มากขึ้น เราจำเป็นต้องมี state visibility และ completion condition ที่ตรวจสอบได้ และเมื่อ agent เข้าถึง browser, external site หรือ production tool ได้มากขึ้น safety boundary ต้องย้ายจากข้อความเตือนใน prompt ไปอยู่ที่ permission rule, classifier, sandbox, hook และ review gate

ขอบเขตการอัปเดตและหลักการคัดกรอง

เพื่อไม่ให้เนื้อหาซ้ำกับบทเดิม บทนี้ใช้หลักการคัดกรองดังต่อไปนี้:

- พื้นฐาน Context Engineering และ Context Compression ให้อ้างกลับไปยังบทที่ 1
- พื้นฐาน Claude Code, permission และ Plan mode ให้อ้างกลับไปยังบทที่ 2 และบทที่ 6
- พื้นฐาน MCP, plugin และ marketplace ให้อ้างกลับไปยังบทที่ 4
- พื้นฐาน GitHub automation ให้อ้างกลับไปยังบทที่ 5
- พื้นฐาน custom subagent และ parallel orchestration ให้อ้างกลับไปยังบทที่ 7
- พื้นฐาน Output styles และ status line ให้อ้างกลับไปยังบทที่ 8
- พื้นฐาน Agent Skills ให้อ้างกลับไปยังบทที่ 9
- พื้นฐาน Desktop, cloud agent และ Git worktree ให้อ้างกลับไปยังบทที่ 10

ดังนั้น หัวข้ออย่าง background subagent ในบทนี้จะกล่าวเฉพาะ behavior ใหม่ที่ทำงานแบบ background โดย default, การส่ง permission prompt กลับมาที่ main session และ nested delegation เท่านั้น เราจะไม่สร้าง subagent definition ซ้ำตั้งแต่ต้น เช่นเดียวกับ browser automation ที่จะเปรียบเทียบ trust boundary ของ Chrome, Desktop Browser และ Computer use โดยไม่ทำ tutorial Playwright MCP ซ้ำจากบทที่ 2

> คำว่า **background agent** ในบทที่ 10 หมายถึง session ที่ทำงานแยกจาก terminal หรือทำงานบน cloud ส่วน **background subagent** ในบทนี้หมายถึง worker ภายใน session เดียวกัน ทั้งสองอย่างทำงานพร้อมกันได้ แต่มี lifecycle, context และ isolation ต่างกัน นอกจากนี้ `/agents` และ `claude agents` เป็นคนละ interface: `/agents` ใน version ปัจจุบันให้คำแนะนำเรื่องการสร้าง subagent definition ส่วน `claude agents` เปิด Agent view สำหรับจัดการ session หลายรายการ

จาก feature list สู่ operating model

operating model ที่เหมาะกับ Claude Code รุ่นปัจจุบันประกอบด้วย loop ต่อไปนี้:

1. **กำหนด outcome** ด้วย prompt, spec หรือ `/goal`
2. **แยกงาน** ด้วย subagent, Agent view หรือ Dynamic workflow ตาม scale
3. **จำกัด authority** ด้วย permission mode, deny/ask rule, sandbox และ managed policy
4. **สังเกต runtime** ด้วย Agent view, `/tasks`, Monitor, `/usage` และ session recap
5. **ตรวจสอบผลลัพธ์** ด้วย test, Browser, Computer use, `/code-review` และ security review
6. **สื่อสารผลลัพธ์** ด้วย pull request, Artifact หรือ onboarding guide

แนวทางนี้สำคัญกว่าการจำ command ทีละรายการ เพราะ command ใหม่แต่ละตัวมีตำแหน่งใน control loop เดียวกัน ตัวอย่างเช่น `/goal` เพิ่ม persistence ให้ execution แต่ไม่ได้แทน spec, test หรือ review ส่วน Auto mode ลด permission interruption แต่ไม่ได้แทน sandbox และ permission deny rule

model และ Context Engineering รุ่นปัจจุบัน

Claude Sonnet 5, context 1M token และ adaptive thinking

Week 27 เปิดตัว **Claude Sonnet 5** เป็น default model สำหรับ Pro, Team Standard และ Enterprise subscription seat บางประเภท โดยต้องใช้ Claude Code v2.1.197 ขึ้นไป จุดเปลี่ยนที่สำคัญต่อ coding workflow คือ native context window ขนาด 1M token และ adaptive thinking ที่เปิดโดย default ตามรายละเอียดใน Week 27 digest และ Model configuration

เลือก model ได้ด้วย command ต่อไปนี้:

```
/model claude-sonnet-5
```

หรือใช้ alias ซึ่งชี้ไปยัง Sonnet รุ่นที่ provider แนะนำ:

```
/model sonnet
```

context 1M token ลดแรงกดดันจาก context ceiling ใน session ขนาดใหญ่ แต่ไม่ได้ยกเลิกหลัก Context Engineering จากบทที่ 1 เหตุผลมี 3 ประการ:

- context ที่ยาวยังมี cost และ latency
- log, tool output และ duplicate instruction ยังทำให้ signal-to-noise ratio ลดลง
- subagent, filesystem artifact และ Context Compression ยังคงช่วยแยก concern และลด context pollution

ดังนั้น ควรมอง context 1M token เป็น headroom สำหรับงานที่จำเป็นต้องรักษา dependency ระยะยาว ไม่ใช่เหตุผลให้โหลดทั้ง repository, documentation และ transcript ทุกอย่างเข้าสู่ main context โดยไม่มีการคัดกรอง

adaptive thinking ทำให้ model ปรับ reasoning effort ตามความยากของ task ได้ดีขึ้น แต่คุณยังควบคุม effort ได้ด้วย `/effort` ตัวเลือกที่ปรากฏจริงขึ้นอยู่กับ model, plan และ provider ตัวอย่างเช่น:

```
/effort high
```

สำหรับ task ที่ต้องการการวิเคราะห์อย่างลึกมาก สามารถเลือกระดับที่ model รองรับได้จาก interface `/effort` แทนการ hardcode ค่าเดียวใน project instruction วิธีนี้ช่วยให้ codebase ใช้งานได้แม้ default model เปลี่ยนในอนาคต

ออกแบบ model policy โดยไม่ผูก workflow กับ model เดียว

model alias มีประโยชน์สำหรับ personal workflow เพราะได้รับ upgrade ตาม provider แต่ production automation ต้องพิจารณาความสามารถในการทำซ้ำ หาก test baseline หรือ regulated workflow ต้องการ model version ที่แน่นอน ให้ pin full model ID ใน configuration ที่เหมาะสม และแยก policy ของ main session, subagent และ organization ออกจากกัน

Week 24 เพิ่ม `fallbackModel` แบบ chain ได้สูงสุด 3 model เมื่อ primary model ไม่พร้อมใช้งานหรือ overloaded ตัวอย่าง user settings มีดังนี้:

```
{
  "model": "claude-sonnet-5",
  "fallbackModel": [
    "claude-opus-4-8",
    "claude-haiku-4-5"
  ]
}
```

หรือกำหนดเฉพาะ session จาก shell:

```
claude --fallback-model sonnet,haiku
```

fallback chain มีไว้รักษา availability ไม่ใช่รับประกันว่า output จากทุก model จะเหมือนกัน authentication error, billing error, rate limit, request-size error และ transport error ไม่ทำให้ระบบสลับ model ตาม chain นี้ และ model ที่อยู่นอก organization allowlist จะถูกข้าม

แนวปฏิบัติที่เหมาะสมคือใช้ test, schema, acceptance criteria และ review gate เป็น contract ของ workflow แทนการสมมุติว่า model ทุกตัวจะ reasoning เหมือนกัน นี่คือการย้าย reliability จาก model identity ไปยัง system design

orchestration ของงานระยะยาว

บทที่ 7 อธิบาย custom subagent และ parallel execution ส่วนบทที่ 10 อธิบาย Desktop, cloud agent และ worktree แล้ว การอัปเดตรุ่นปัจจุบันเพิ่ม control plane สำหรับงานหลายระดับ ตั้งแต่ worker ภายใน session ไปจนถึง session จำนวนมากและ workflow ที่สร้าง orchestration script เอง

Agent view และ background session

`claude agents` เปิด **Agent view** ซึ่งรวม session ที่กำลังทำงาน รอ input และเสร็จแล้วไว้ในหน้าจอเดียว feature นี้เป็น research preview และต้องใช้ v2.1.139 ขึ้นไป คุณสามารถ attach เข้า session ใด session หนึ่งเพื่ออ่าน conversation เต็ม แล้วกลับไปยังรายการรวมได้

เปิด Agent view จาก shell:

```
claude agents
```

Agent view เหมาะเมื่อทีมมีหลาย task ที่เป็นอิสระ เช่น bug fix, flaky test investigation และ pull request review แต่ละ task มี context, lifecycle และ working directory ของตนเอง Week 28 ปรับ row ให้แสดง state และ headline ที่อ่านง่ายขึ้น พร้อมเชื่อม pull request ที่ session แก้ไขหรือส่งขึ้น remote ได้ดีขึ้น รายละเอียดล่าสุดอยู่ใน Agent view guide

สิ่งสำคัญคือ background session ใน Agent view ทำงานบนเครื่อง local และใช้ subscription quota ตามจำนวน session การเรียก 10 session พร้อมกันจึงไม่ได้ทำให้ resource ฟรีขึ้น แต่เปลี่ยน latency แบบลำดับให้เป็น throughput แบบ parallel คุณควรจำกัด concurrency ตาม dependency ของงาน, test infrastructure และ review capacity ของทีม

กำหนด task ให้ Agent view ได้จาก prompt area หรือ dispatch จาก shell แล้วใช้ state ต่อไปนี้เป็น operating signal:

- **Working**: session ยังดำเนินการอยู่ ไม่ควร interrupt โดยไม่มีเหตุผล
- **Needs input**: session ต้องการ decision หรือ permission จากมนุษย์
- **Completed**: ตรวจ artifact, diff, test และ PR ก่อนปิด session

session ที่ pin ไว้จะคง process ให้พร้อมใช้งานขณะ idle ส่วน background session ปรากฏใน `/resume` พร้อม marker `bg` ทำให้สามารถกลับไปทำงานต่อโดยไม่ต้องพึ่ง terminal เดิม

background subagent และ nested delegation

ตั้งแต่ v2.1.198 subagent จะทำงานแบบ background โดย default เมื่อ main session ไม่จำเป็นต้องรอผลทันที Claude จึงทำงานส่วนอื่นต่อได้และรับผลเมื่อ subagent เสร็จ หาก main session ต้องใช้ผลก่อนดำเนินขั้นต่อไป ระบบยังเลือก foreground execution ได้

คุณสามารถ pin behavior ของ custom subagent ด้วย front matter:

```
---
name: dependency-auditor
description: Audits dependency changes and reports compatibility risks
tools: Read, Grep, Glob, Bash
model: sonnet
background: true
---
Review dependency changes. Return only confirmed risks with file references.
```

ตั้งแต่ v2.1.186 permission prompt ของ background subagent จะปรากฏใน main session แทนการ auto-deny และตั้งแต่ v2.1.172 subagent สามารถ spawn nested subagent ได้ลึกสูงสุด 5 ระดับ ความสามารถนี้เหมาะกับ task ที่มี hierarchy ตามธรรมชาติ เช่น migration coordinator ที่มอบหมาย package audit ให้ worker หลายตัว

อย่างไรก็ตาม nested delegation ไม่ควรเป็น default สำหรับทุก task เพราะแต่ละระดับเพิ่ม token usage, state coordination และโอกาสเกิด duplicate work ก่อนเปิดให้ subagent spawn worker เพิ่ม ควรกำหนด output contract และ ownership ของ file หรือ module ให้ชัดเจน

> ใช้ **background subagent** เมื่อ side task อยู่ภายใน context ของ session หลัก ใช้ **Agent view** เมื่อ task ต้องมี session lifecycle ของตนเอง และใช้ **Dynamic workflow** เมื่อ scale ใหญ่เกินกว่าที่ conversation เดียวจะประสาน dependency ทั้งหมดได้อย่างน่าเชื่อถือ

ใช้ /goal กับ completion condition ที่ตรวจสอบได้

`/goal` เพิ่ม persistence ให้ Claude ทำงานต่อหลาย turn จน completion condition เป็นจริง หลังแต่ละ turn evaluator จะตรวจ condition หากยังไม่ผ่าน ระบบจะเริ่ม turn ถัดไปโดยไม่ต้องรอ prompt ใหม่จากผู้ใช้ feature นี้ทำงานใน interactive session, `-p`, Desktop และ Remote Control ตาม Goals guide

ตัวอย่างที่ดี:

```
/goal all tests in test/auth pass and npm run lint exits with code 0
```

ตัวอย่างที่ไม่ดี:

```
/goal make the authentication better
```

condition แรกมี oracle ที่ตรวจได้ ส่วน condition ที่สองเปิดช่องให้ agent ตีความคำว่า “better” เอง `/goal` จึงทำงานได้ดีที่สุดเมื่อ condition ผูกกับ test, build, lint, metric หรือ artifact ที่ตรวจสอบได้

ดู goal ปัจจุบันด้วย:

```
/goal
```

ยกเลิกก่อนสำเร็จด้วย:

```
/goal clear
```

evaluator ของ `/goal` ไม่เรียก command หรืออ่าน file ด้วยตนเอง แต่ตัดสินจากหลักฐานที่ Claude แสดงไว้ใน conversation เท่านั้น ดังนั้น condition ต้องกำหนดให้ Claude เรียก test หรือ verification command และรายงานผลลง transcript อย่างชัดเจน

`/goal` ไม่ได้แทน Plan mode หรือ spec โดย Plan mode ใช้กำหนด approach ก่อนแก้ code ขณะที่ `/goal` ใช้รักษา execution หลังเริ่มลงมือ คุณสามารถใช้ทั้งสองร่วมกัน: วางแผนและอนุมัติก่อน จากนั้นตั้ง measurable goal สำหรับ implementation และ verification

Dynamic workflows และ ultracode สำหรับงานระดับ codebase

Dynamic workflows เปิดตัวเป็น research preview ใน Week 22 ต้องใช้ Claude Code v2.1.154 ขึ้นไปและใช้ได้บน paid plan โดยผู้ใช้ Pro ต้องเปิด Dynamic workflows จาก `/config` feature นี้ให้ Claude เขียน JavaScript orchestration script สำหรับ task แล้วกระจายงานไปยัง subagent จำนวนมากใน background เหมาะกับ codebase-wide audit, migration ขนาดใหญ่ หรือ research ที่ต้อง cross-check หลายแหล่ง จัดการ run ด้วย `/workflows`

ตัวอย่าง prompt:

```
Create a workflow that migrates every internal fetch() call
to the new HttpClient wrapper, runs package-level tests,
and reports any call site that cannot be migrated safely.
```

trigger keyword ที่ระบุชัดในรุ่นปัจจุบันคือ `ultracode` แม้ natural-language request ที่ขอ workflow ยังใช้ได้ ตัวอย่างเช่น:

```
Use ultracode to audit every API route for missing authorization checks.
```

ใน version ที่รองรับ สามารถตั้ง effort เป็น `ultracode` เพื่อรวม xhigh reasoning กับ automatic workflow orchestration:

```
/effort ultracode
```

>_ agentic-ebook-th.vercel.app

อยากอ่านต่อจนจบใช่ไหม?

คุณเพิ่งดูช่วงต้นของบทที่ 1, 2, 3 และ 12

ฉบับเต็ม 12 บท 306 หน้า ครอบคลุมตั้งแต่ Context Engineering,

MCP, subagent, multi-agent workflow ไปจนถึง production-ready operations

฿1,199

ราคาเปิดตัว (ลดจาก ฿1,499)

สั่งซื้อและดาวน์โหลดได้ทันทีที่

agentic-ebook-th.vercel.app

ซื้อครั้งเดียว ดาวน์โหลดได้ไม่จำกัด · ได้ไฟล์ฉบับล่าสุดเสมอ