BOOK THREE IN THE PROMPT ENGINEERING SERIES

The Era of the Custom Agent Harness

Why Pi Changes Everything for Beginners

Darryl Williams

Blockcheckbook

AN ADVANCED EXTENSION ON AGENT ARCHITECTURE AND CUSTOM HARNESS DESIGN

Foreword: The Harness Revolution

From Features to Frameworks

In the first two books of this series, we explored the art and science of prompt engineering. We learned how to structure our requests for maximum accuracy, and then how to infuse those requests with emotional intelligence and contextual resonance. We mastered the "what" and the "how" of speaking to AI.

But there was a missing piece. We were still thinking of AI as a conversation partner — something we talk to, wait for, and then act upon. We were the harness. The human was the loop.

That changes now.

The custom agent harness represents the most significant paradigm shift in AI development since the large language model itself. It is the recognition that AI is not merely a conversational tool but an autonomous worker — one that can observe, act, learn, and iterate on our behalf while we sleep, while we think, while we direct.

Pi, the minimal agent harness at the center of this revolution, embodies a philosophy that is radical in its simplicity: the AI tool should adapt to your workflow, not the other way around. In an era of ever-expanding AI platforms trying to lock you into their ecosystems, Pi offers something different — a foundation you can own, extend, and make entirely your own.

This book is for beginners who sense that AI could be doing more for them. It is for the curious who have watched AI demos and wondered, "But how does that actually work?" It is for builders who want to move from asking AI questions to deploying AI workers.

The era of one-size-fits-all AI is ending. The future belongs to harnesses you can adapt, extend, and own.

Welcome to the revolution.

Dedication

To every beginner who has stared at a chat interface and thought, "There must be more than this." To the builders who refuse to accept that AI's greatest potential ends at the send button. And to the open-source community that proves, every single day, that the most powerful technology is the technology you can shape with your own hands.

Table of Contents

INTRODUCTION

Beyond the Chat Window

Let me ask you something that might sting a little: while you were asleep last night, did your AI tool actually move your business forward?

Did it respond to a prospect? Chase down an estimate you sent? Schedule even one appointment?

No? Then you don't need more AI features. You need a harness.

We have entered a new era — the era of the custom agent harness. And for beginners, this shift is nothing short of revolutionary. For years, we have been handed "AI features" wrapped in shiny interfaces, asked to adapt our workflows to their rigid patterns. But the most powerful shift in AI development is happening quietly: the rise of the harness — the framework that governs how an AI agent thinks, acts, and learns.

This book builds upon the foundations established in our first two volumes. If Book One taught you how to speak to AI with precision, and Book Two taught you how to speak with resonance, then Book Three teaches you how to build AI that works for you — continuously, autonomously, and under your control.

The Three Layers of AI Mastery

Human-AI interaction operates on multiple layers, all of which must be addressed for true mastery:

1. **The Prompt Layer (What you say):** This is the foundational skill — crafting instructions that are clear, structured, and logically sound. Book One covered this comprehensively.
2. **The Resonance Layer (How it feels):** This involves encoding emotion, tone, and cultural context into your prompts so the output connects with human audiences. Book Two explored this art form.
3. **The Harness Layer (What it does):** This is the domain of this book. It encompasses the infrastructure that turns AI from a conversational tool into an autonomous worker — one that can read files, write code, run commands, remember context, and keep working until the job is done.

A prompt engineered for a harness guides the AI to master that third, crucial layer. It ensures the final output is not just a collection of correct facts or emotionally resonant prose, but the result of a persistent, evolving conversation with your work.

What This Book Is (and Isn't)

This book is a practical guide to understanding, deploying, and customizing agent harnesses, with a specific focus on Pi — the minimal agent harness that has become the standard for beginners who want to move beyond chat.

This book is *not* a programming manual. You do not need to be a software engineer to read it. You do not need to understand how large language models are trained. What you need is curiosity, a willingness to use your terminal, and the belief that AI should work for you — not the other way around.

> *"The era of one-size-fits-all AI is ending. The future belongs to harnesses you can adapt, extend, and own."*

By the end of this book, you will understand the architecture of an agent harness, be comfortable navigating Pi's core systems, and have a clear roadmap for building your own extensions. You will be ready to move from asking AI questions to deploying AI workers.

PART I

The Philosophy of the Agent Harness

Understanding the architecture that turns AI from a chatbot into a worker

CHAPTER 1

Defining the Agent Harness and Its Components

1.1 What Is an Agent Harness?

Think of an AI model — like ChatGPT, Claude, or Gemini — as a brilliant brain. It knows things. It can reason, synthesize, and generate. But it cannot *do* anything on its own.

The harness is the body that gives that brain hands and eyes. It is the invisible engine that turns a smart chat into a productive worker. Without a harness, the model is a philosopher trapped in a void — brilliant but powerless. With a harness, it becomes an architect who can draft blueprints, a developer who can write and execute code, a researcher who can search and compile.

Every coding agent ships with two things: a model that reasons and a harness that does everything else. The harness determines:

- **Visibility:** What the model can see — which tools are available, which files exist, what the current state of the project is.
- **Capability:** What the model can do — read files, write code, run bash commands, query databases, call APIs.
- **Memory:** What the model remembers across conversations — session history, project context, previous decisions.
- **Boundaries:** When the model stops — preventing runaway costs, infinite loops, or unauthorized actions.
- **Environment:** Where the model runs — safely sandboxed in the cloud, on your local machine, or in a container.

1.2 The Model and the Harness: A Perfect Partnership

To understand why the harness matters so much, consider the relationship between model and harness as analogous to the relationship between a pilot and an aircraft.

The pilot (the model) makes decisions. They choose altitude, heading, speed. They respond to weather conditions and air traffic. But without the aircraft (the harness), the pilot is simply a person with good instincts and nowhere to go. The aircraft provides the engines, the control surfaces, the instruments, the fuel system, and the safety mechanisms.

A great pilot in a terrible aircraft will have a terrible flight. A mediocre pilot in an exceptional aircraft will still get to their destination. The harness, in many ways, is the great equalizer.

Component	Pilot (Model)	Aircraft (Harness)
Decision-making	Reasoning, planning, judgment	Execution, tool calling, state management
Input	Reads prompts and context	Reads files, captures output, monitors environment
Output	Generates text and plans	Writes files, runs commands, updates systems
Safety	Follows guidelines	Enforces timeouts, limits costs, scopes permissions

1.3 Why Beginners Need Harness Literacy

If you are new to AI development, you might reasonably ask: "Why should I care about the harness? Isn't the model the interesting part?"

The answer is that harness literacy is what separates AI tourists from AI builders. Tourists visit AI through web interfaces. They ask questions, get answers, and leave. Builders deploy AI through harnesses. They set up systems that work continuously, integrate with their tools, and improve over time.

Understanding the harness gives you:

- **Control:** You decide what the AI can and cannot do. You define the boundaries.
- **Trust:** You understand why the AI behaves the way it does. There are no black boxes.
- **Flexibility:** You can adapt the harness to your workflow, not the other way around.
- **Ownership:** The harness runs on your machine, under your control, with your data.

> **Key Concept: The Harness as Interface**
>
> The harness is not merely a technical implementation — it is the interface through which human intention becomes machine action. A well-designed harness makes the AI feel like an extension of your own capabilities. A poorly designed harness makes the AI feel like an obstacle to getting things done.

CHAPTER 2

The Observe-Act-Learn-Iterate Loop

2.1 Understanding the Agent Loop

At the heart of every agent harness is a loop — the continuous cycle that makes AI agents actually useful. This loop is the fundamental mechanism by which an agent transforms from a passive responder into an active worker. Understanding this loop is essential to understanding how Pi, and agent harnesses in general, operate.

The classical agent loop follows a simple pattern:

1. **Observe:** The agent perceives its environment — reading files, checking system state, receiving user input.
2. **Act:** The agent takes action based on its observations — writing code, running commands, making API calls.
3. **Learn:** The agent processes the results of its actions — understanding output, identifying errors, recognizing patterns.
4. **Iterate:** The agent repeats the cycle, incorporating what it learned into its next observation and action.

This loop is not merely a technical detail. It is the reason agents feel different from chatbots. A chatbot completes a request and waits. An agent completes a request, observes the result, takes follow-up action, and keeps going until the task is truly done.

2.2 The Pi Loop: Intake to Persistence

In Pi (which uses OpenClaw under the hood), the loop follows a clear, structured flow that beginners can understand and experienced users can extend:

1 **Intake** — Receive your request. This is where the loop begins, with a clear statement of what needs to be accomplished.

2 **Context Assembly** — Gather relevant information and history. Pi pulls from the session tree, previous conversations, and available files to build a comprehensive understanding of the current state.

3. **Model Inference** — The model thinks and decides what to do. This is where the "brain" makes its plan, choosing which tools to call and in what order.

4. **Tool Execution** — Actions are performed. Files are read or written. Commands are executed. The harness translates the model's decisions into real-world effects.

5. **Streaming Reply** — The response comes back in real-time. You see what the agent is doing as it happens, not after it finishes.

6. **Persistence** — Everything is saved for context in future interactions. The session tree grows. The memory expands. The agent becomes more capable with each loop.

This loop repeats continuously — observe what happened, act on it, learn from the result, and iterate. It is not about checking items off a list. It is about creating a persistent, evolving conversation with your work.

2.3 From "To-Do List" to "Till-Done List"

This is where Pi fundamentally changes the game for beginners.

Most AI tools operate like a glorified to-do list. You ask, they answer, and you are left to figure out the rest. The conversation ends when the model generates its response. The work of implementation, verification, and iteration falls entirely on you.

But a proper agent harness works like a "till-done" system — it keeps working until the job is actually finished. If the code has a syntax error, the agent sees it, fixes it, and tries again. If a file is missing, the agent locates it or creates it. If a command fails, the agent reads the error message and adjusts its approach.

> *"The difference between a chatbot and an agent is the difference between a consultant who gives advice and a colleague who stays late to get it done."*

Characteristic	Traditional AI (To-Do List)	Agent Harness (Till-Done List)
Interaction Model	Request → Response	Goal → Loop → Completion
Error Handling	User must identify and fix	Agent self-corrects
Context	Per-conversation only	Persistent across sessions
User Role	Active participant in every step	Director and approver
Outcome	Information delivered	Work completed

PART II

The Pi Architecture

Inside the systems that make Pi the minimal harness with maximum potential

CHAPTER 3

The Session Tree — Never Lose Context

3.1 Sessions as Branching Trees

Sessions in Pi are not linear chats. They are stored as branching trees. Each entry has an ID and a parent ID, meaning the conversation structure mirrors the way humans actually think — non-linear, exploratory, with frequent diversions and returns.

In a traditional chat interface, if you want to explore a different direction, you have two choices: start a completely new conversation and lose all context, or continue in the current thread and hope the model can keep track. Neither option is satisfactory.

Pi's session tree solves this elegantly. Every message exists as a node in a tree. Every node knows its parent. This means you can:

- Navigate backward to any previous point in the conversation and continue from there, with full context preserved.
- Fork the conversation at any point, exploring a new direction without affecting the original branch.
- Clone an active branch into an entirely new session, preserving the context while giving yourself a clean slate.

All history is preserved in a single file. You never lose where you have been, and you can always explore new paths without starting over.

3.2 Navigation Commands: /tree, /fork, /clone

Pi provides three essential commands for navigating the session tree:

Command	Purpose	When to Use
`/tree`	Display the full session tree and navigate to any node	When you want to return to a previous point in the conversation
`/fork`	Create a new branch from the current node	When you want to explore an alternative approach
`/clone`	Copy the current branch into a new session	When you want to start fresh while preserving context

These commands transform the session from a static transcript into a living workspace. You are no longer bound to the linear flow of conversation. You can explore, backtrack, branch, and merge your way to the right solution.

> **Key Concept: The Session Tree as Memory Palace**
>
> Think of the session tree as a memory palace for your AI agent. Just as a memory palace allows a human to store and retrieve vast amounts of information by associating it with physical locations, the session tree allows an agent to store and retrieve vast amounts of context by associating it with navigable positions in the conversation history.

CHAPTER 4

The Message Queue — Steer While It Works

4.1 Queued Steering Messages

Here is where Pi feels genuinely magical. While the agent is working, you can steer it in real-time.

Traditional AI systems operate in a strict turn-based model. You send a message. You wait. The system responds. Then you can send another message. There is no opportunity to intervene, correct, or redirect while the system is processing.

Pi breaks this model with queued steering messages. While the agent is working — reading files, writing code, running commands — you can press Enter to queue a message that will be delivered after the current turn finishes its tool calls.

This means you are not sitting around waiting. You are actively steering the agent while it works — just like directing a capable colleague who is skilled enough to work independently but smart enough to take direction.

The steering message might be:

- "Wait, don't use that library — use this one instead."
- "Actually, let's focus on the authentication module first."
- "That approach looks good, but can you add error handling?"

4.2 Queued Follow-Up Messages

In addition to steering messages, Pi supports queued follow-up messages (sent with Alt+Enter). These are delivered only after all work is complete — meaning you can queue up your next request while the agent is still finishing the current one.

The distinction is important:

Message Type	Trigger	Delivery	Use Case
Steering	Enter	After current turn's tool calls	Redirect or correct mid-task
Follow-Up	Alt+Enter	After all work is complete	Queue next task

This queuing system transforms the user experience from passive waiting to active collaboration. You become the conductor of an orchestra that actually plays while you direct it.

CHAPTER 5

Context Compaction — Never Forget the Important Stuff

5.1 How Compaction Works

Long sessions can exhaust context windows. Every large language model has a limit on how much text it can process at once — typically measured in "tokens," where a token is roughly a word or part of a word. When a conversation becomes too long, the model starts to "forget" the beginning of the conversation, or the system has to truncate it.

Pi handles this through a process called **context compaction** — summarizing older messages while keeping recent ones fresh and fully detailed. The compaction process analyzes the session history, identifies key decisions, important facts, and critical context, and creates a compressed summary that preserves what matters while removing what does not.

The result is that Pi can maintain effective conversations far longer than traditional chat interfaces. Even when the raw conversation history would exceed the model's context window, the compacted version fits comfortably, ensuring the agent always has the context it needs.

5.2 Manual and Automatic Triggers

Pi offers two ways to trigger compaction:

- **Manual compaction** — Triggered with the `/compact` command. This gives you full control over when compaction happens. Use it when you notice the agent's responses slowing down or when you want to clean up before starting a complex new task.
- **Automatic compaction** — Triggered by the system when the context window approaches its limit. This ensures you never hit a hard cutoff unexpectedly.

Importantly, the full history remains available in the JSONL file. You can always revisit it with `/tree`. Nothing is truly lost — compaction is an optimization for the model's working memory, not a deletion of your conversation history.

Key Concept: Compaction as Note-Taking

Think of compaction as the difference between having a detailed transcript of every meeting you have ever attended versus having well-organized notes of the key decisions and action items. The transcript preserves everything but is overwhelming. The notes preserve what matters and are immediately useful. Compaction gives your agent the notes it needs without the burden of the full transcript.

CHAPTER 6

Four Tools, Infinite Possibilities

6.1 The Default Toolset

Here is what makes Pi special for beginners: Pi is a minimal agent harness you can adapt to your workflows, not the other way around. And its minimalism starts with its default toolset.

By default, Pi gives the model just four tools:

Tool	Function	Example Use
`read`	Read files	Read source code, configuration files, documentation
`write`	Create files	Create new scripts, write output files, generate reports
`edit`	Modify existing files	Update code, fix bugs, refactor, add features
`bash`	Run terminal commands	Install packages, run tests, build projects, check system state

That is it. Four tools. But with those four tools, the model can fulfill nearly any request related to software development, system administration, data processing, and technical writing. The simplicity is the point — Pi does not try to anticipate every use case. It gives you a foundation powerful enough to build anything.

6.2 Building Beyond the Basics

When you need more than the default four tools, you have several options:

- **Ask Pi to build what you want.** The most powerful feature of Pi is that Pi can extend itself. You can ask Pi to create new tools, add new commands, or integrate with external services — and Pi will write the code to do it.
- **Install third-party Pi packages.** The Pi ecosystem includes community-contributed packages that add tools, commands, and integrations. These can be installed without modifying Pi's core code.
- **Create your own TypeScript extensions.** For more advanced customization, you can write TypeScript extensions that define new tools with custom behavior.

- **Define reusable skills in Markdown.** Skills are markdown files that teach Pi how to perform specific tasks. They are portable, shareable, and easy to create.
- **Create prompt templates for common tasks.** Templates allow you to save frequently used prompt patterns and apply them with a single command.

This extensibility model means Pi grows with you. When you are a beginner, the four default tools are enough. As you gain experience, you add capabilities. The harness becomes more powerful not by bloating its core but by letting you attach exactly what you need.

PART III

Security, Building & The Future

Deploying harnesses responsibly and building toward an autonomous future

CHAPTER 7

Security and Privacy — Your Harness, Your Rules

7.1 Identity and Scoping

The era of the custom harness brings serious responsibility. As MIT Technology Review recently outlined, securing agentic systems requires treating agents like powerful, semi-autonomous users with strict boundaries.

Every agent should run as a non-human principal with permissions constrained to specific roles and tasks. In the Pi ecosystem, you can scope what each agent can access through explicit tool definitions and project trust settings.

This means:

- An agent working on a frontend project should not have access to backend credentials.
- An agent generating documentation should not have permission to modify source code.
- An agent running in a testing environment should not be able to affect production systems.

7.2 The "Everything Agent" Trap

One of the most dangerous anti-patterns in agent development is building a single agent that does everything. This "everything agent" has access to all tools, all files, all systems, and all credentials. It is a security disaster waiting to happen.

Pi's minimal approach encourages a better pattern by default: give agents the fewest tools they need to accomplish their goal. If an agent only needs to read files and write documentation, it should only have the `read` and `write` tools. If it needs to run tests, add `bash`. But do not give it tools it does not need.

> *"No everything agents. An agent should have a sharply defined purpose, a carefully scoped set of tools, and clear boundaries on what it can touch."*

Anti-Pattern	Better Approach
Single agent with all tools	Multiple specialized agents with scoped tools
Unrestricted file system access	Workspace isolation with explicit allow-lists
Shared credentials across tasks	Task-specific, time-limited access tokens
No audit trail	Every action logged and reviewable

7.3 Circuit Breakers and Explicit Interfaces

Agents need hard limits on time, money, and API calls. Without these limits, a misbehaving agent can run indefinitely, generating enormous costs or causing system damage.

Pi enforces these at the infrastructure layer — the agent cannot talk itself out of them. Configurable timeouts prevent runaway sessions. API call limits prevent cost explosions. And explicit interfaces ensure that when an agent talks to an external system, it does so through well-defined, observable channels.

Key security mechanisms include:

- **Timeouts:** Maximum duration for any single operation or session.
- **Rate limiting:** Maximum number of API calls per minute or hour.
- **Cost caps:** Maximum spend per session, day, or project.
- **Sandboxing:** Execution environments isolated from the host system.
- **Audit logging:** Every tool call recorded with timestamps and parameters.

CHAPTER 8

Learning to Build — Your First Week With Pi

8.1 Day 1: Installation and First Conversation

If you are a beginner, your first week with Pi should be about exploration and comfort. Do not try to build anything complex on day one. Just talk to it.

Installation is straightforward:

```
npm install -g --ignore-scripts @earendil-works/pi-coding-agent
export ANTHROPIC_API_KEY=sk-ant-...
pi
```

Once Pi is running, start simple:

- Ask it to read a file in your project: "Read the README.md file and tell me what this project does."
- Ask it to write a small script: "Write a Python script that counts the number of files in this directory."
- Ask it to run a command: "Run `ls -la` and show me the results."

Get comfortable with the four tools. Watch how the agent thinks, plans, and executes. Observe the loop in action.

8.2 Days 2-3: Mastering the Commands

Once you are comfortable with basic interaction, master the essential commands that give you control over the harness:

Command	Purpose
`/model`	Switch between AI models — useful for comparing outputs or using different models for different tasks
`/tree`	Navigate session history — jump to any point in the conversation
`/compact`	Manage context — manually trigger context compaction
`/settings`	Configure behavior — adjust timeouts, tool permissions, and other settings

8.3 Days 4-5: Your First Extension

Pi uses TypeScript extensions. Start simple — add a custom command or tool that fits your specific workflow. For example:

- A command that runs your project's test suite and formats the output nicely.
- A tool that searches your documentation for specific keywords.
- A command that generates a commit message based on your recent changes.

The key is to identify a repetitive task in your workflow and automate it. This is where Pi begins to pay dividends — not in grand projects, but in the elimination of small friction.

8.4 Days 6-7: Creating Your First Package

Package your extensions, skills, and prompt templates into something you can share or reuse across projects. A Pi package is simply a collection of related extensions bundled together with metadata.

Creating a package involves:

1. Organizing your extensions into a directory structure.
2. Creating a `package.json` or equivalent manifest file.
3. Writing documentation explaining what the package does and how to use it.
4. Sharing it with your team or the community (optional).

By the end of your first week, you should have a Pi setup that is noticeably more capable than the default installation — customized to your workflow, your projects, and your preferences.

CHAPTER 9

Extending Without Forking

9.1 Ask Pi to Build What You Want

Pi ships with powerful defaults but skips bloated features like built-in sub-agents and plan mode. Instead of giving you everything, Pi gives you the ability to build anything. And the most powerful way to extend Pi is to simply ask Pi to build the extension for you.

This recursive capability — using Pi to improve Pi — is one of its most elegant design features. You do not need to read the source code or understand the extension API in depth. You describe what you want, and Pi writes the extension.

For example:

- "Create a tool that fetches the latest weather for a given city using the OpenWeatherMap API."
- "Add a command that generates a changelog from recent git commits."
- "Build a skill that helps me write better unit tests by analyzing my code and suggesting edge cases."

9.2 Third-Party Packages and TypeScript Extensions

Beyond asking Pi to build extensions, you can install extensions created by others. The Pi ecosystem supports third-party packages that can be installed and configured without modifying Pi's core code.

For more advanced customization, you can write your own TypeScript extensions. These give you full control over the behavior of new tools, including:

- Custom parameter validation and error handling.
- Integration with external APIs and services.
- Complex multi-step workflows that chain multiple actions together.
- Custom output formatting and presentation.

9.3 Skills and Prompt Templates

Skills are one of Pi's most innovative extensibility mechanisms. A skill is a Markdown file that teaches Pi how to perform a specific task. Unlike code extensions, skills require no programming — just clear writing.

A typical skill file includes:

- A description of what the skill does.
- Step-by-step instructions for performing the task.
- Examples of good and bad outputs.
- Any relevant context or constraints.

Prompt templates, similarly, are reusable prompt patterns saved as files. They allow you to standardize common interactions — code reviews, documentation generation, refactoring requests — so you do not have to write the same instructions repeatedly.

Extension Type	Skill Level	Use Case
Ask Pi to build	Beginner	Quick customizations
Skills (Markdown)	Beginner	Reusable task instructions
Prompt templates	Beginner	Standardized prompt patterns
Third-party packages	Intermediate	Community extensions
TypeScript extensions	Advanced	Custom tools and integrations

CHAPTER 10

Multiple Providers, One Interface

10.1 Subscription-Based Providers

One of Pi's most practical features is its ability to work with multiple AI providers through a single, consistent interface. This means you are not locked into one model or one company. You can use the best model for each task, or simply use the one you already have access to.

Pi supports subscription-based access through:

- **Anthropic Claude Pro/Max** — Industry-leading reasoning and code generation.
- **OpenAI ChatGPT Plus/Pro (Codex)** — Broad general knowledge and strong tool use.
- **GitHub Copilot** — Deep integration with development workflows.

10.2 API Key Providers

For developers who prefer API keys, Pi supports virtually every major provider:

- Anthropic (Claude)
- OpenAI (GPT-4, GPT-3.5)
- Azure OpenAI
- DeepSeek
- Google Gemini
- Mistral
- Groq
- Cerebras
- xAI
- OpenRouter
- Hugging Face
- Fireworks
- Together AI

10.3 Adding Custom Providers

You can even add custom providers that speak a supported API format. This is particularly useful for:

- Self-hosted models running on your own infrastructure.
- Enterprise models deployed internally.
- Experimental models not yet supported officially.
- Specialized models fine-tuned for specific domains.

> **Key Concept: Model as Replaceable Engine**
>
> Think of the model provider as the engine in your car. Pi is the chassis, the steering, the brakes, and the instruments. You can swap the engine without changing how you drive. This means you can start with a free or low-cost model and upgrade to more powerful models as your needs grow — all without changing your workflow.

CHAPTER 11

The Future Is Yours to Build

We stand at an inflection point in the history of human-AI collaboration. The first wave of AI tools gave us access to knowledge. The second wave gave us the ability to generate content. The third wave — the wave we are entering now — gives us autonomous workers that can observe, act, learn, and iterate on our behalf.

The custom agent harness is the foundation of this third wave. It is the infrastructure that turns AI from a tool you use into a worker you deploy. And Pi represents the most accessible entry point into this new paradigm.

Pi is built on a simple philosophy: the AI tool should adapt to your workflow, not the other way around. It gives you the minimal foundation and the full power to build exactly what you need. No lock-in. No bloat. Just a harness you can make your own.

As you move forward from this book, remember these principles:

- **Start minimal.** Use the four default tools until you genuinely need more. Complexity should be earned, not assumed.
- **Scope aggressively.** Give each agent the fewest tools it needs. Security and simplicity go hand in hand.
- **Extend through use.** The best extensions come from real needs discovered through real work, not from imagining what you might someday need.
- **Steer actively.** The message queue is your superpower. Use it. The best agent interactions are collaborations, not handoffs.
- **Preserve context.** Use the session tree. Branch, fork, and clone your way to better solutions without losing where you have been.

The era of one-size-fits-all AI is ending. The future belongs to harnesses you can adapt, extend, and own.

That is the revolution. Not more features. More control.

The question is not whether your AI tool can do more. The question is whether it can do your work — the way you need it done.

Pi can. And it will.

APPENDIX A

Glossary of Agent Harness Terms

- **Agent:** An AI system that can autonomously observe its environment, make decisions, take actions, and learn from results.
- **Agent Harness:** The framework that governs how an AI agent thinks, acts, and learns — including tool availability, memory, boundaries, and execution environment.
- **API Key:** A credential used to authenticate requests to an AI provider's API.
- **Bash Tool:** One of Pi's four default tools; allows the agent to execute terminal commands.
- **Circuit Breaker:** A safety mechanism that stops an agent when it exceeds defined limits on time, cost, or API calls.
- **Context Compaction:** The process of summarizing older conversation history to fit within a model's context window while preserving key information.
- **Context Window:** The maximum amount of text a language model can process in a single request.
- **Edit Tool:** One of Pi's four default tools; allows the agent to modify existing files.
- **Everything Agent:** An anti-pattern where a single agent is given unrestricted access to all tools and systems.
- **Extension:** A code module that adds new capabilities to Pi, typically written in TypeScript.
- **Fork:** Creating a new branch in the session tree from the current node.
- **Intake:** The first stage of the Pi loop; receiving the user's request.
- **JSONL:** JSON Lines format; the file format used by Pi to store session history.
- **Loop:** The continuous cycle of Observe, Act, Learn, and Iterate that drives agent behavior.
- **Message Queue:** Pi's system for allowing users to send steering and follow-up messages while the agent is working.
- **Model Inference:** The stage where the AI model processes context and decides what actions to take.
- **Model Provider:** A company or service that hosts and serves AI models (e.g., Anthropic, OpenAI).
- **Package:** A bundle of related Pi extensions, skills, and templates that can be installed and shared.
- **Read Tool:** One of Pi's four default tools; allows the agent to read files.

- **Session Tree:** Pi's branching conversation structure where each message is a node with a parent ID.
- **Skill:** A Markdown file that teaches Pi how to perform a specific task.
- **Steering Message:** A message queued during agent execution that redirects or corrects the agent's approach.
- **Tool Execution:** The stage where the harness carries out the actions decided by the model.
- **Write Tool:** One of Pi's four default tools; allows the agent to create new files.

APPENDIX B

Command Reference and Quick Start Guide

Quick Start

Install Pi:

```
curl -fsSL https://pi.dev/install.sh | sh
```

Or install via npm:

```
npm install -g --ignore-scripts @earendil-works/pi-coding-agent
```

Set your API key:

```
export ANTHROPIC_API_KEY=sk-ant-...
```

Launch Pi:

```
pi
```

Essential Commands

Command	Description
`/model`	Switch between AI models
`/tree`	Display and navigate the session tree
`/fork`	Create a new branch from current node
`/clone`	Copy current branch to a new session
`/compact`	Manually trigger context compaction
`/settings`	Configure Pi behavior and preferences

Default Tools

Tool	Function
`read`	Read file contents
`write`	Create new files
`edit`	Modify existing files
`bash`	Execute terminal commands

Supported Providers

Subscriptions: Anthropic Claude Pro/Max, OpenAI ChatGPT Plus/Pro (Codex), GitHub Copilot

API Keys: Anthropic, OpenAI, Azure OpenAI, DeepSeek, Google Gemini, Mistral, Groq, Cerebras, xAI, OpenRouter, Hugging Face, Fireworks, Together AI

Resources

- **GitHub:** openclaw/openclaw for SDK integration examples
- **Documentation:** Full docs at pi.dev

CONCLUSION

The Harness Mindset

The era of the custom agent harness is not just a technological shift. It is a mindset shift. It represents a transition from thinking about AI as something you talk to, toward thinking about AI as something you deploy. From AI as oracle to AI as worker. From AI as interface to AI as infrastructure.

This mindset has profound implications for how we work, build, and create. When you adopt the harness mindset:

- **You stop asking "Can AI do this?"** and start asking "How do I harness AI to do this?"
- **You stop accepting tool limitations** and start extending tools to match your needs.
- **You stop treating AI as a vendor relationship** and start treating it as a buildable, ownable capability.

Pi embodies this mindset in its purest form. It is minimal by design, extensible by philosophy, and yours by default. It does not try to be everything. It tries to be the perfect foundation — and then gets out of your way.

The three books in this series form a complete progression. Book One taught you precision — how to craft prompts that get accurate results. Book Two taught you resonance — how to craft prompts that connect with human audiences. Book Three teaches you architecture — how to build systems where AI works continuously, autonomously, and under your control.

Together, they represent the full spectrum of modern prompt engineering: from the precision of logic, through the warmth of resonance, to the power of autonomous action.

The future belongs to those who can master all three.

FINAL THOUGHTS

A Pledge of Responsible Harness Building

Mastery of agent harnesses carries a heavy ethical burden. The ability to deploy autonomous AI workers — systems that can read your files, run commands, and make changes without constant supervision — is a profound power. As practitioners of this advanced craft, we must hold ourselves to a higher standard.

Consider this a pledge:

1. **I will use agent harnesses to augment human capability, not to replace human judgment.** Autonomous does not mean unsupervised. I will maintain oversight of the agents I deploy.
2. **I will scope my agents responsibly.** I will give each agent the minimum tools and permissions it needs, and no more. I will not build "everything agents."
3. **I will respect privacy and security.** I will not deploy agents that access data or systems without proper authorization. I will secure my API keys and credentials.
4. **I will be transparent about AI involvement.** When agents perform work that affects others, I will disclose that involvement appropriately.
5. **I will use my skills to build, create, and solve problems — never to deceive, exploit, or harm.**

Continue to explore. Continue to experiment. Continue to build. The future of AI is not just in achieving the next logical breakthrough or generating the most resonant prose. It is in building harnesses that empower us to do more than we ever could alone.

That is the promise of the custom agent harness. That is the future Pi makes accessible.

Build it well.

The Era of the Custom Agent Harness: Why Pi Changes Everything for Beginners

Darryl Williams

Blockcheckbook

The third book in the Prompt Engineering Series:
Book One: Understanding Prompts — From Beginners to Experts
Book Two: Vibe Coding — Mastering AI Resonance and Emotional Intelligence
Book Three: The Era of the Custom Agent Harness

Blockcheckbook is dedicated to publishing cutting-edge resources on the intersection of technology, human creativity, and advanced AI methodologies.
We believe that the most powerful tools are those that enhance our humanity, not replace it.

www.blockcheckbook.com