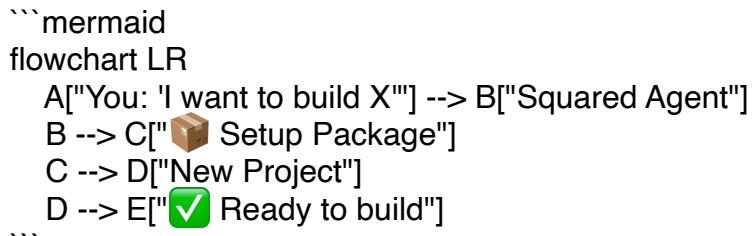


Squared Agent

Bootstrap new projects with Claude Code. Ship faster with built-in workflows.

Stop spending the first hour of every project configuring tools, setting up git workflows, and remembering which commands work best. Squared Agent packages everything you need — session management, branch protection, tool intelligence, and platform-specific guides — into a single setup you copy to new projects.



The Problem

Every new project starts with the same friction:

I Problem	I What Happens I
----- -----	
I **Manual setup**	I Spend 30+ minutes configuring Claude Code, plugins, permissions, and commands I
I **Messy main branches**	I Accidental commits to main → merge conflicts → lost work I
I **Credential leakage**	I MCP API keys accidentally committed or shared between projects I
I **Lost learnings**	I Discover a great tool shortcut, forget it by next project I
I **No workflow**	I Every session starts cold — no context, no plan I
I **No cost visibility**	I Token usage scattered across sessions → no idea what you're spending I

Squared Agent solves all of these.

Quick Start

1. Get the Agent

```
```bash
git clone https://github.com/squared-lemons/squared-agent.git
````
```

```
cd squared-agent
```

```
---
```

2. Open with Claude Code

```
```bash
```

```
claude .
```

```

```

### ### 3. Start a New Idea

```

```

```
/new-idea
```

```

```

Have a discovery conversation:

- Describe what you want to build
- Discuss requirements and platform options
- Make technical decisions together
- Claude generates a complete project package

### ### 4. Copy to Your Project

A folder opens with your setup package. Copy its contents to your new project folder.

### ### 5. Run the Setup

In your new project folder:

```
```bash
```

```
claude .
```

```
---
```

Tell Claude: "Read SETUP.md and help me set up this project"

The agent handles the rest — configures plugins, creates commands, sets up permissions, and gets you ready to build.

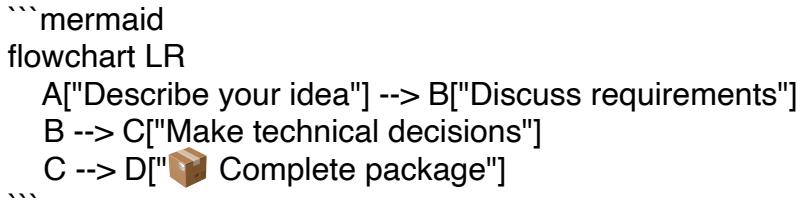
```
---
```

Creating Setup Packages

Two ways to create setup packages for new projects:

`/new-idea` — Discovery Conversation

Best for new projects where you're still figuring out requirements.



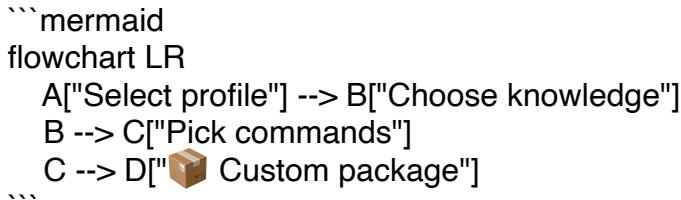
The conversation covers:

- What you're building and who it's for
- Platform options (web, mobile, desktop)
- Technical decisions with tradeoffs explained
- Scope for v1 vs future features

Output: `PROJECT-BRIEF.md`, `TECHNICAL-DECISIONS.md`, `SETUP.md`, plus relevant knowledge and commands.

`/prepare-setup` — Component Selection

Best when you know what you need and want to pick specific components.



Select from:

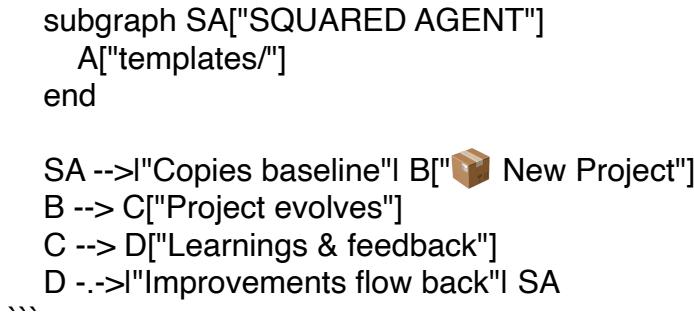
- **Profiles** — Base configurations (developer workflow, permissions, hooks)
- **Knowledge** — Platform guides (Next.js, etc.)
- **Commands** — Workflow guides (end-session, new-feature, etc.)
- **Tasks** — One-time setup activities (codebase investigation)

[Full templates reference →](templates/README.md)

What Your Projects Inherit

Everything below defines the baseline every spawned project receives. Squared Agent runs this setup, then the new project evolves independently — building its own tool intelligence, capturing its own learnings, and feeding improvements back.





Session Git Workflow

****This agent uses the [Session Git Workflow](templates/workflows/Session-Git-Workflow.md)** — and so does every project it spawns.**

| Command | When to Use |

----- -----		
`/start-session` Beginning of work — checks branch safety, loads context		
`/new-feature "desc"` Starting new work — creates feature branch or worktree		
`/commit` During work — quick commit with approval		
`/complete-feature` Feature is done — merge to main or create PR		
`/end-session` Done for now — update docs, capture learnings, commit		

Protected branches (`main`, `master`, `develop`, `release/*`) block direct changes and guide you to create a feature branch first.

****Token tracking**** is built in — `/end-session` captures usage, `/summary` calculates costs. Track spending against subscription limits and know when to upgrade or optimize.

****[Full Session Git Workflow →](templates/workflows/Session-Git-Workflow.md)****

Tools & Integrations

Squared Agent works with 40+ tools organized across four categories.

MCP Servers via Toolhive

We recommend [Toolhive](https://github.com/stacklok/toolhive) for managing MCP servers. Here are the plugins we suggest configuring:

| Server | Key Tools | Purpose |

----- ----- -----		

I **GitHub** I `search_repositories`, `search_code`, `list_issues`, `create_issue` I GitHub API integration |
I **Perplexity** I `perplexity_research`, `perplexity_ask`, `perplexity_reason` I AI-powered web search |
I **FireCrawl** I `firecrawl_scrape`, `firecrawl_crawl`, `firecrawl_map` I Web scraping and crawling |
I **ShadCN** I `get_component`, `list_components` I UI component library |
I **Context7** I `resolve-library-id`, `query-docs` I Live documentation lookup |
I **DataForSeo** I SEO analysis, keyword research I Search engine optimization |
I **n8n** I Workflow automation I No-code automation |
I **Playwright** I Browser automation, screenshots I E2E testing |

> **MCP (Model Context Protocol)**: A standard for connecting AI models to external tools and services. Think of it as plugins for Claude.

Claude Code Plugins

Pre-configured plugins that add specialized capabilities:

Plugin	Command	What It Does
I **feature-dev**	I `/feature-dev`	I Architecture-first feature planning with code-explorer, code-architect, and code-reviewer agents
I **ralph-loop**	I `/ralph-loop`	I Autonomous implement → test → iterate loop until task is complete
I **frontend-design**	I `/frontend-design`	I Production-grade UI that avoids generic AI aesthetics
I **code-simplifier**	I —	I Refines code for clarity while preserving functionality
I **context7**	I —	I Fetches up-to-date library documentation
I **playwright**	I —	I Browser automation and visual testing

Browser Automation

Via `claude-in-chrome` MCP server:

Tool	Purpose
I `tabs_context_mcp`	I Get available browser tabs
I `read_page`	I Accessibility tree of page elements
I `find`	I Natural language element search
I `form_input`	I Fill form fields
I `navigate`	I Go to URLs, back/forward
I `computer`	I Click, type, scroll, screenshot

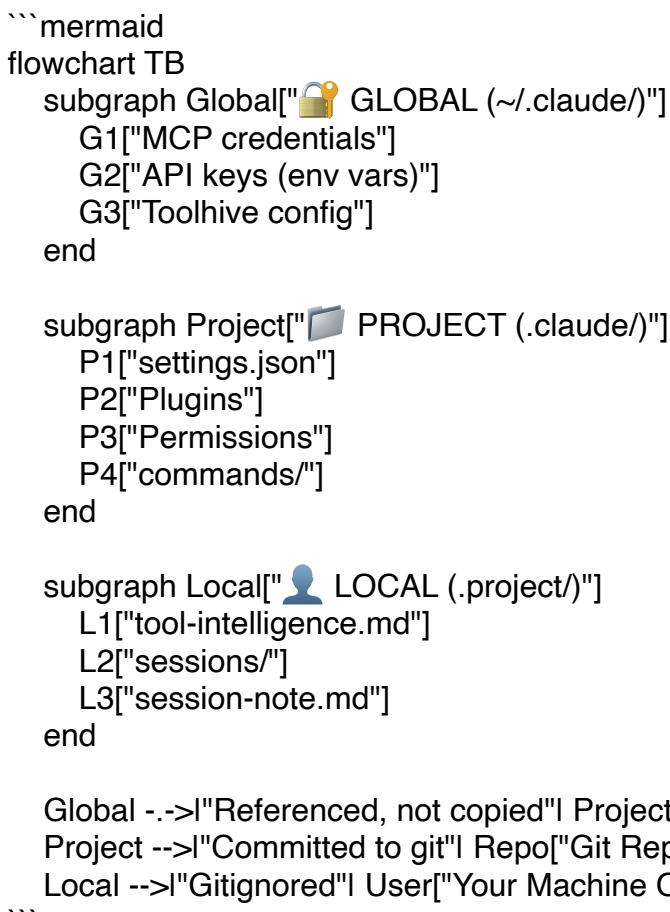
Core Tools

Built into Claude Code:

Tool	Purpose
`Glob`	Fast file pattern matching
`Grep`	Content search with regex
`Read`	Read files (including images, PDFs)
`Write`	Create new files
`Edit`	Modify existing files
`Bash`	Run terminal commands
`Task`	Launch specialized agents

MCP Security Model

Credentials never leak between projects. Squared Agent uses a three-layer separation:



Layer	Location	Contains	In Git?	
Global	~/.claude/	MCP credentials, API keys	No	

| **Project** | `.claude/` | Plugins, permissions, commands | Yes |
| **Local** | `.project/` | Tool intelligence, session logs | No |

Why This Matters

- **Templates are credential-free**: Copy setup packages without exposing secrets
 - **Each project is isolated**: Tool intelligence learned in one project stays there
 - **MCP servers via environment**: Credentials live in your shell, not in code

—

Tool Intelligence

The agent learns which tools work best for which tasks — and keeps a work log for reporting.

```

```mermaid
graph TD
 flowchart TB
 Session["DURING SESSION"]
 Session --> A["/start-session"]
 A --> B["tool-intelligence.md"]
 B --> C["session-note.md"]
 C --> D["Work with optimal tools"]
 D --> E["Capture session"]
 E --> F["Session log
(local, not in repo)"]
 E --> G["Tool intelligence
(local, not in repo)"]
 E --> H["Session note
(local, not in repo)"]
 Reporting["/summary"]
 F --> I["Generate report"]
 I --> J["Today / Week / Month"]
 J --> K["Accomplishments
ready to share"]
```

```

What Gets Captured

I Output I Location I In Repo? I Purpose I

| **Session logs** | `.project/sessions/` | No (gitignored) | Timestamped work history + token usage |

| **Tool intelligence** | `.project/tool-intelligence.md` | No (gitignored) | Learned
shortcuts and patterns |

```
| **Token usage** | `project/token-usage.md` | No (gitignored) | Cumulative cost  
tracking |  
| **Session note** | `project/session-note.md` | No (gitignored) | Task handoff for  
next session |
```

What Tool Intelligence Tracks

- **Toolhive shortcuts**: Which MCP servers you use most
- **Plugin patterns**: How `/feature-dev` uses `/ralph-loop` where appropriate
- **Browser tips**: Effective `claude-in-chrome` patterns
- **Core efficiency**: When to use Task agents vs direct tools

Token Usage & Cost Tracking

Every session captures raw token usage from Claude Code:

| Metric | Description |
|--------------------|--|
| ----- | ----- |
| **Billing type** | `subscription` (Claude Code plan) or `api` (background agents) |
| **Input tokens** | Tokens in your prompts |
| **Output tokens** | Tokens in Claude's responses |
| **Cache read** | Tokens retrieved from cache (cheaper) |
| **Cache creation** | Tokens added to cache |

Costs and limits are calculated at report time:

- **Subscription sessions**: Tracked against your configured daily/hourly limits to assess tier needs
- **API sessions**: Charged per token (estimated in `/summary` reports)

Configure your subscription limits in `project/token-usage.md` to track usage against your plan's daily and hourly caps. `/summary` will show % utilization and recommend tier changes if you're frequently hitting limits.

Reporting with `/summary`

Generate accomplishments reports from your session logs:

...

```
/summary today    # What you did today  
/summary week     # This week's work  
/summary month    # Monthly accomplishments  
...
```

Output includes:

- Categorized git commits (features, fixes, refactors, etc.)
- Session highlights from logs
- **Token usage** by billing type (subscription vs API)

- **Subscription limit analysis** with % utilization and tier recommendations
- **Estimated API costs** calculated at report time with current pricing
- Cache efficiency percentage

Copy-paste ready for standups, status updates, or client reports.

How It Works

1. `/start-session` loads tool preferences and previous session note
2. Claude proactively selects appropriate tools without you asking
3. `/end-session` saves session log + updates tool intelligence + leaves note for next time
4. `/summary` pulls from session logs to generate reports
5. Each session starts smarter than the last

All data stays local in ` `.project/` (gitignored). Personal to each user, compounds over time.

Commands Reference

Session & Git

| Command | Description |
|-----------------------|--|
| ----- ----- | |
| `/start-session` | Begin session with branch awareness and context loading |
| `/new-feature "desc"` | Create feature branch (or worktree) for safe development |
| `/complete-feature` | Wrap up feature branch — merge or create PR |
| `/end-session` | End session, update docs, capture learnings, commit |
| `/commit` | Draft commit message, get approval, commit |

Project Creation

| Command | Description |
|------------------|---|
| ----- ----- | |
| `/new-idea` | Discovery conversation → complete project package |
| `/prepare-setup` | Create generic setup package with selected components |

Utilities

| Command | Description |
|-----------------|--|
| ----- ----- | |
| `/summary` | Generate accomplishments report from git history |
| `/how-to-use` | Display the human-editable guide |
| `/list-tools` | List all commands, plugins, and tools |
| `/get-feedback` | Process inbox and implement improvements |

Project Structure

```
templates/      # Content copied to new projects
  commands/     # Command implementation guides
  knowledge/    # Framework guides (Next.js, etc.)
  ux-guides/    # UI/UX patterns
  profiles/     # Setup profiles (developer/, etc.)
  tasks/        # One-time setup tasks

inbox/          # Ideas and feedback for improvements
  ideas/         # Your ideas to discuss
  from-projects/ # Feedback from spawned projects

suggestions/   # Agent proposals (categorized)
  knowledge/    # Proposed new guides
  commands/     # Proposed command improvements
  workflow/     # Proposed workflow changes

docs/           # Documentation
.claude/        # Claude Code configuration
  commands/     # Active slash commands

.project/       # Local data (gitignored)
  sessions/     # Session logs by date
  tool-intelligence.md # Learned tool preferences
  token-usage.md # Cumulative token stats
---
```

Continuous Improvement

Every project you spawn can teach Squared Agent something new.

```
```mermaid
graph TD
 subgraph Input ["FEEDBACK SOURCES"]
 A["inbox/ideas/
Your ideas"]
 B["inbox/from-projects/
Project feedback"]
 end

 subgraph Process ["PROCESSING"]
 C["/get-feedback"]
 end

 A --> C
 B --> C
```

```

D["Discuss & plan"]
E["Implement"]
end

subgraph Output["RESULTS"]
F["templates/ improved"]
G["LEARNINGS.md patterns"]
end

A --> C
B --> C
C --> D
D --> E
E --> F
E --> G
F -.->"Better projects"! B
...

```

### ### The Feedback Loop

1. \*\*During project work\*\*: `/end-session` generates creator feedback
  2. \*\*Save feedback\*\*: Copy it to `inbox/from-projects/` in this repo
  3. \*\*Process feedback\*\*: Run `/get-feedback` to review and implement
  4. \*\*Templates improve\*\*: Future projects benefit from past learnings
- 

### ## Documentation

I Document	I What's Inside I
----- -----	
I [templates/README.md](templates/README.md)	I Full templates reference — workflows, profiles, knowledge, commands, tasks I
I [docs/workflow.md](docs/workflow.md)	I Development workflow and best practices I
I [docs/commands.md](docs/commands.md)	I Full command documentation I
I [docs/plugins.md](docs/plugins.md)	I Plugin configuration details I
I [docs/content.md](docs/content.md)	I Available profiles, knowledge, and tasks I
I [docs/feedback.md](docs/feedback.md)	I Creator feedback loop I
I [docs/how-to-use.md](docs/how-to-use.md)	I Human-editable quick start guide I

---

### ## License

Private — Squared Lemons