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.

```mermaid
flowchart LR
    A["You: 'I want to build X'"] --> B["Squared Agent"]
    B --> C["📦 Setup Package"]
    C --> D["New Project"]
    D --> E["✅ Ready to build"]
```

The Problem

Every new project starts with the same friction:

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

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.

```mermaid
flowchart LR
    A["Describe your idea"] --> B["Discuss requirements"]
    B --> C["Make technical decisions"]
    C --> D["📦 Complete package"]
```

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.

```mermaid
flowchart LR
    A["Select profile"] --> B["Choose knowledge"]
    B --> C["Pick commands"]
    C --> D["📦 Custom package"]
```

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.

```mermaid
flowchart LR
```

```
    subgraph SA["SQUARED AGENT"]
        A["templates/"]
    end

    SA -->|"Copies baseline"| B["📦 New Project"]
    B --> C["Project evolves"]
    C --> D["Learnings & feedback"]
    D -.->|"Improvements flow back"| SA
```

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

GitHub	`search_repositories`, `search_code`, `list_issues`, `create_issue`	GitHub API integration
Perplexity	`perplexity_research`, `perplexity_ask`, `perplexity_reason`	AI-powered web search
FireCrawl	`firecrawl_scrape`, `firecrawl_crawl`, `firecrawl_map`	Web scraping and crawling
ShadCN	`get_component`, `list_components`	UI component library
Context7	`resolve-library-id`, `query-docs`	Live documentation lookup
DataForSeo	SEO analysis, keyword research	Search engine optimization
n8n	Workflow automation	No-code automation
Playwright	Browser automation, screenshots	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
feature-dev	`/feature-dev`	Architecture-first feature planning with code-explorer, code-architect, and code-reviewer agents
ralph-loop	`/ralph-loop`	Autonomous implement → test → iterate loop until task is complete
frontend-design	`/frontend-design`	Production-grade UI that avoids generic AI aesthetics
code-simplifier	—	Refines code for clarity while preserving functionality
context7	—	Fetches up-to-date library documentation
playwright	—	Browser automation and visual testing

Browser Automation

Via `claude-in-chrome` MCP server:

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

```mermaid
flowchart TB
    subgraph Global["🔐 GLOBAL (~/.claude/)"]
        G1["MCP credentials"]
        G2["API keys (env vars)"]
        G3["Toolhive config"]
    end

    subgraph Project["📁 PROJECT (.claude/)"]
        P1["settings.json"]
        P2["Plugins"]
        P3["Permissions"]
        P4["commands/"]
    end

    subgraph Local["👤 LOCAL (.project/)"]
        L1["tool-intelligence.md"]
        L2["sessions/"]
        L3["session-note.md"]
    end

    Global -.->|"Referenced, not copied"| Project
    Project -->|"Committed to git"| Repo["Git Repository"]
    Local -->|"Gitignored"| User["Your Machine Only"]
```

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
flowchart TB
    subgraph Session["DURING SESSION"]
        A["/start-session"] -->|"Loads"| B["tool-intelligence.md"]
        A -->|"Reads"| C["session-note.md"]
        B --> D["Work with optimal tools"]
        C --> D
    end

    subgraph EndSession["/end-session"]
        D --> E["Capture session"]
        E --> F["📝 Session log<br/>(local, not in repo)"]
        E --> G["🧠 Tool intelligence<br/>(local, not in repo)"]
        E --> H["📋 Session note<br/>(local, not in repo)"]
    end

    subgraph Reporting["/summary"]
        F -->|"Reads logs"| I["Generate report"]
        I --> J["Today / Week / Month"]
        J --> K["📊 Accomplishments<br/>ready to share"]
    end
```

What Gets Captured

Output	Location	In Repo?	Purpose
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
flowchart TB
    subgraph Input["FEEDBACK SOURCES"]
        A["inbox/ideas/<br/>Your ideas"]
        B["inbox/from-projects/<br/>Project feedback"]
    end

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

```
        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

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

License