The Séance: Ghost County's Complete Development Ritual

The Séance: Ghost County’s Complete Development Ritual

Overview

The **séance** is Ghost County's complete idea-to-shipped workflow - a guided ritual that takes you from raw concept to completed, archived work with zero manual coordination. It's the full SDLC in one command.

The Three-Part Ritual

``` 🔮 Phase 1: Scrying (Planning) → /seance –scry or –plan 👻 Phase 2: Summoning (Execution) → /seance –summon or –execute 🌾 Phase 3: Reaping (Archival) → /seance –reap or –archive ```

Each phase can be run independently or as part of the complete ritual.

Entry Points

Interactive Mode

```bash /seance ```

You get a choice: - **[A] Add something new** → Runs scrying (planning) phase - **[B] Summon the spirits** → Runs summoning (execution) phase - **Tell Claude what to do** → Custom input

Direct Mode (with an idea)

```bash /seance "Add dark mode to the app" ``` Immediately starts scrying with your idea, then offers to summon.

Explicit Phases
```

```bash /seance –scry # or –plan (just planning) /seance –summon # or –execute (just execution) /seance –reap # or –archive (just archival) ```

Mystical vs Normie: - **Mystical:** `–scry`, `–summon`, `–reap` (embrace the occult vibes) - **Normie:** `–plan`, `–execute`, `–archive` (keep it practical)

🔮 Phase 1: Scrying (Planning)

Purpose: Transform raw ideas into formal, actionable roadmaps.

Planning Depth Modes

Scrying supports three depth levels that control how thoroughly requirements are analyzed:

Mode	Flags	When to Use	Output
Quick	`--quick`	Simple features, bug fixes, well-understood work	Basic requirements + roadmap (skip strategic analysis)
Medium	(default)	Standard features requiring planning	Requirements + light analysis + roadmap
Deep	`--deep`	Complex features, new products, strategic initiatives	Full requirements + comprehensive analysis (JTBD, Kano, RICE) + detailed roadmap

Examples: ```bash /seance –quick “Fix login button styling” # Quick mode /seance “Add user profile page” # Medium mode (default) /seance –deep “Build OAuth integration system” # Deep mode ```
```

What Happens

The Project Manager agent takes your raw idea and runs it through the Ghost County planning gauntlet:

1. **Requirements Development**
    - Applies 14-dimension rubric (atomic, testable, estimated, independent)
    - Creates formal requirements document
    - Defines acceptance criteria with "Definition of Done"
2. **Strategic Analysis**
    - **JTBD (Jobs to be Done):** Why does this matter to users?
    - **Kano Analysis:** Is this a delight, performance need, or table stakes?
    - **RICE Scoring:** What's the impact? (Reach × Impact × Confidence / Effort)
3. **Roadmap Creation**
    - Breaks work into S/M sized chunks (completable in one session)
    - Assigns to appropriate agents (Dev-Backend, Dev-Frontend, Dev-Infrastructure, etc.)
    - Organizes into batches with dependency mapping
    - Outputs to `.haunt/plans/roadmap.md`

Output

A formal, actionable roadmap with REQ-XXX items: ```markdown ### ⚪ REQ-087: Implement OAuth provider integration **Effort:** S (2 hours) **Complexity:** MODERATE **Agent:** Dev-Backend **Completion:** OAuth tokens generated and validated **Blocked by:** None ```

When to Use

- New feature requests
- Bug reports that need planning
- Vague "we should build X" ideas
- Multi-component features requiring breakdown

👻 Phase 2: Summoning (Execution)

Purpose: Spawn autonomous agents to work through the roadmap.

What Happens

The séance reads your roadmap and orchestrates an agent swarm:

1. **Parse Roadmap**
   - Finds all ⚪ Not Started items
   - Finds all 🟡 In Progress items (unfinished work)
   - Respects "Blocked by:" dependencies
   - Groups work into parallel batches
2. **Spawn Agent Swarm**
   - Launches appropriate agents in parallel (where dependencies allow)
   - Each agent gets its specific REQ-XXX assignment
   - Agents work autonomously:
     - Read requirements and acceptance criteria
     - Follow TDD workflow (tests first, then implementation)
     - Run tests to verify completion
     - Commit code with proper conventions
3. **Continuous Work**
   - Agents work until their requirement is 🟢 Complete
   - All tasks checked off: `- [x]`
   - All tests passing
   - Code committed with REQ-XXX reference

Output

- Parallel agent execution working through the roadmap
- Features implemented, tested, and committed
- Requirements marked 🟢 Complete when done

When to Use

- After scrying creates a roadmap
- When roadmap has unstarted (⚪) or in-progress (🟡) items
- To resume work on an existing roadmap

- For batch execution of multiple requirements

🌾 Phase 3: Reaping (Archival)

Purpose: Archive completed work and clean the roadmap.

What Happens

After agents complete their work, the séance automatically reaps the harvest:

1. **Verification** (7-Point Checklist)

   Before archiving any 🟢 Complete requirement, verification ensures:

   - ✅ All task checkboxes marked `[x]` (not `[ ]`)
   - ✅ Completion criteria met (from requirement's "Completion:" field)
   - ✅ Tests passing (appropriate test command for work type)
   - ✅ Files modified as specified (from requirement's "Files:" field)
   - ✅ Documentation updated (if applicable)
   - ✅ Security review completed (if code involves user input, auth, databases, external APIs, file operations, or dependencies)
   - ✅ Self-validation performed (agent re-read requirement, reviewed own code, confirmed tests actually test the feature, checked against known anti-patterns)

2. **Archival**

   - Moves completed requirements to `.haunt/completed/roadmap-archive.md`
   - Preserves implementation notes and completion dates
   - Removes from active roadmap
   - Keeps roadmap under 500 lines for performance

3. **Completion Report**

   - Shows what was accomplished
   - Reports any incomplete items
   - Highlights verification issues
   - Suggests next steps

Output

- Clean roadmap with only active work
- Archived history in `.haunt/completed/`
- Completion summary report

When to Use

- After summoning phase completes
- When roadmap has many 🟢 Complete items (>10)
- To clean up before starting new work
- Weekly/sprint-end cleanup ritual

Complete Workflow Example

Scenario: You want to add OAuth login support.

Step 1: Scrying

```bash $ /seance “Add OAuth login support” ```

What happens: ``` 🔮 Scrying the future...

Analyzing requirements... Running strategic analysis (JTBD, Kano, RICE)... Creating roadmap...

✅ Requirements developed: - REQ-087: Implement OAuth provider integration (S, 2hr) - REQ-088: Add login redirect flow (S, 1.5hr) - REQ-089: Secure token storage (S, 2hr) - REQ-090: Add logout endpoint (XS, 30min) - REQ-091: Update UI with OAuth buttons (S, 1hr)

Strategic analysis complete. Estimated 8-12 hours total work. Organized into 2 batches with dependencies.

Roadmap created at .haunt/plans/roadmap.md

Ready to summon the spirits? ```
```

Step 2: Summoning

You: “yes” (or `/seance –summon`)

What happens: ``` 👻 The spirits rise...

Summoning gco-dev-backend for REQ-087... Summoning gco-dev-backend for REQ-088... Summoning gco-dev-infrastructure for REQ-089...

[Agents work in parallel, running tests, committing code...] [8-12 hours of autonomous work...]

Agent gco-dev-backend completed REQ-087 ✓ Agent gco-dev-backend completed REQ-088 ✓ Agent gco-dev-infrastructure completed REQ-089 ✓

[Batch 2 dependencies cleared, continuing...]

Summoning gco-dev-backend for REQ-090... Summoning gco-dev-frontend for REQ-091...

Agent gco-dev-backend completed REQ-090 ✓ Agent gco-dev-frontend completed REQ-091 ✓

All requirements complete. ```

Step 3: Reaping

Automatic after summoning completes (or `/seance –reap`)

What happens: ``` 🌾 Reaping the harvest...

Verifying completion... ✓ All tasks checked off for REQ-087, REQ-088, REQ-089, REQ-090, REQ-091 ✓ All completion criteria met ✓ All tests passing ✓ All commits created

Archiving to .haunt/completed/roadmap-archive.md...

🌙 The harvest is complete.

Completed and Archived: - 🟢 REQ-087: Implement OAuth provider integration - 🟢 REQ-088: Add login redirect flow - 🟢 REQ-089: Secure token storage - 🟢 REQ-090: Add logout endpoint - 🟢 REQ-091: Update UI with OAuth buttons

5 requirements archived Active roadmap cleaned OAuth login feature complete ✓ ```

Result: OAuth login shipped, tested, committed, documented, and archived. Zero manual coordination.

Why This Matters

Without the Séance

- You write requirements manually
- You spawn agents one by one
- You track which agents finished
- You manually archive completed work
- You context-switch constantly
- Coordination overhead scales with complexity

With the Séance

- Say what you want (one line or interactive choice)
- The séance handles: requirements → analysis → roadmap → execution → archival
- You come back to shipped features
- Zero coordination overhead
- Full SDLC automation

Advanced Usage

Partial Ritual (Individual Phases)

Just planning: ```bash /seance –scry “Add rate limiting to API” # Creates roadmap, doesn’t execute ```

Just execution: ```bash /seance –summon # Executes existing roadmap items ```

Just cleanup: ```bash /seance –reap # Archives completed work ```

Full Ritual (All Phases)

```bash # Interactive mode /seance > [A] Add something new > [Enter idea: “Add rate limiting”] > [Scrying happens...] > “Ready to summon?” → yes > [Summoning happens...] > [Reaping happens automatically...] ```

Batch Execution

```bash # Create roadmap for multiple features /seance –scry “Feature A” # (adds REQ-101, REQ-102, REQ-103)

/seance –scry “Feature B” # (adds REQ-104, REQ-105)

# Execute everything in parallel

/seance –summon # (spawns 5 agents working in parallel)

# Clean up when done

/seance –reap ```

## Roadmap Sharding (Large Projects)

For projects with many requirements, roadmaps can be **sharded into batch files** to reduce context size and improve performance:

**Monolithic roadmap** (default for most projects): ``` .haunt/plans/roadmap.md ← All requirements in one file ```

**Sharded roadmap** (optimization for 20+ requirements): ``` .haunt/plans/roadmap.md ← Overview + active batch only .haunt/plans/batches/ ├── batch-1-foundation.md ← Completed

Batch 1 ├── batch-2-features.md ← Completed Batch 2 └── batch-3-polish.md ← Active Batch 3 (current work) ```

**How sharding works with séance:** - `roadmap.md` contains the currently active batch plus project overview - Completed batches move to `.haunt/plans/batches/` for archival - Agents load only active batch context (faster startup, less token usage) - Séance automatically detects sharded structure during scrying/summoning

**When to shard:** - Roadmap exceeds 500 lines (performance optimization) - More than 20-30 active requirements - Multi-phase projects with clear batch boundaries

**Benefits:** - Faster agent startup (40-60% less context to load) - Better focus (agents see only relevant work) - Preserves history (completed batches archived, not deleted)

---

# File Outputs

## During Scrying

- `.haunt/plans/roadmap.md` - Main roadmap (active work)
- `.haunt/plans/requirements-document.md` - Formal requirements (if new project)
- `.haunt/plans/requirements-analysis.md` - Strategic analysis (if new project)

## During Summoning

- Source code files (implementation)
- Test files (TDD workflow)
- Git commits (with REQ-XXX references)

## During Reaping

- `.haunt/completed/roadmap-archive.md` - Archived requirements
- Completion reports (summary of what shipped)

---

# Best Practices

## 1. Scry Before You Summon

Don’t skip planning. Even simple features benefit from formal requirements and strategic thinking.

## 2. Reap Regularly

Clean roadmap weekly or after major batches complete. Keeps roadmap under 500 lines for performance.

## 3. Trust the Process

The séance handles coordination. You don’t need to micromanage agents or track progress manually.

## 4. Use Interactive Mode for Discovery

When unsure, use `/seance` (no args) and let the interactive prompts guide you.

## 5. Use Explicit Phases for Control

When you know exactly what you need, use `–scry`, `–summon`, or `–reap` directly.

---

# Troubleshooting

## “No requirements found to summon”

- Run `/seance –scry` first to create a roadmap
- Or check `.haunt/plans/roadmap.md` has ⚪ or 🟡 items

### “Reaping found incomplete tasks”

- Some requirements marked 🟢 but tasks not checked off
- Agent didn’t complete fully - re-summon or fix manually

### “Agent failed during summoning”

- Check error message for specific issue
- Fix blocker, then re-run `/seance –summon`
- Requirements stay 🟡 until completion

### “Roadmap too large (>500 lines)”

- Run `/seance –reap` to archive completed work
- Consider splitting into multiple roadmap files by feature area

---

## The Philosophy

The séance embodies Ghost County’s core principle: **autonomous agents working through formal, testable requirements with zero manual coordination.**

It’s not just a command - it’s a ritual that transforms chaos (vague ideas, bug reports, feature requests) into order (shipped, tested, documented features).

By separating the workflow into three explicit phases (scrying, summoning, reaping), you gain both **control** (run phases independently) and **automation** (let the full ritual handle everything).

Choose your vibe (mystical or practical), but trust the process. The spirits know the way. 🕯️

---

# See Also

- `**/summon** ` - Directly spawn a specific agent (no séance needed)
- `**/banish –all**` - Quick archive command (alias for `/seance –reap`)
- `**Haunt/skills/gco-seance/SKILL.md**` - Technical implementation details
- `**Haunt/skills/gco-project-manager/SKILL.md**` - Planning workflow details
- `**.haunt/plans/roadmap.md**` - Your active roadmap