

Starter Prompt Pack

Six scenarios that show what a knowledge graph changes. Run each against your own codebase in week 2 as an A/B test, then adapt the ones closest to your three committed use cases.

Run every scenario as an A/B test from the start. Install the Orbit A/B Test skill once (Claude Code, one command in your repo):

```
mkdir -p .claude/skills/orbit-ab-test && curl -fsSL -o .claude/skills/orbit-ab-test/SKILL.md https://orbit-dp-program-3fc9a6.gitlab.io/resources/orbit-ab-test-skill.md
```

Then take any prompt below and run `/orbit-ab-test "<the prompt>"`. The skill asks the question twice, once with Orbit and once without, measures time, tool calls, and tokens, and writes the report for you. Drop the "Using GitLab Orbit" opener when you use the skill; it controls tool access itself.

Manual alternative (Duo, or agents without the skill): ask the same question twice, once with Orbit connected and once without, and score each answer on three things: is it **complete** (did it find everything?), is it **grounded** (real files and projects, not guesses?), and is it **fast** (how much digging did the agent do?).

"Using GitLab Orbit" is load bearing. Every prompt below opens with it and ends by asking the agent to show its evidence from the graph. Keep both when you adapt these to your own questions: the opening phrase makes your agent reach for the Orbit tools instead of grepping a checkout, and the evidence ask proves the answer came from the graph. If your agent starts cloning repos or reading files one by one, stop it and re-ask with the Orbit phrase intact.

1. Blast radius

"Using GitLab Orbit, map the blast radius of changing [function or class]. Query the knowledge graph for every definition and reference across our indexed group, not just this repo. For each caller give me the project, file path, and the function it lives in. Finish with a table of affected projects ranked by how many call sites each one has, and tell me how many references the graph returned in total. Answer from the graph: do not clone or grep anything."

Pick a function your team is nervous about changing. **Look for** callers in projects the agent would never have opened on its own, and a total reference count you can sanity-check.

2. Safe deprecation

"Using GitLab Orbit, I am deprecating [endpoint or public method]. Query the graph for every internal consumer across our group: direct callers plus anything that re-exports or wraps them. Group the results by project, and for each consumer output one row I can paste into a migration ticket: project, file path, symbol, and a suggested owner based on recent contributors. Close with the total consumer count from the graph so I know the list is complete."

Pick an endpoint with a real deprecation on your roadmap. **Look for** a caller list complete enough to turn directly into migration tickets, with owners attached.

3. System map

"Using GitLab Orbit, build an onboarding map of [system or service] from the knowledge graph: the main entry points, the most connected files and classes, what it depends on, what depends on it, and the top recent contributors for each area. Present it as a guided tour a new engineer could follow on their first day, with real file paths at every stop. Tell me which parts came from the graph and how many files and definitions it covers."

Pick a system a new hire would struggle with. **Look for** a map an onboarding engineer could actually navigate by, with contributor names for "who to ask". Have your most senior person on that system grade it.

4. Pipeline failure correlation

"Using GitLab Orbit, analyze pipeline health across our whole group from the graph's pipeline data. Which projects and files correlate most with failed pipelines recently? Give me the top 10 with failure counts, the jobs they tend to break, and the merge requests that touched them most recently. Do this from Orbit's pipeline and code graph in one pass, not by opening CI logs project by project, and tell me how many pipelines the analysis covered."

Look for files your platform team already suspects, plus ones they don't. This works across the whole org's pipeline data at once, something no agent can do by reading logs.

5. Codebase health snapshot

"Using GitLab Orbit, give me an executive health snapshot of our group from the knowledge graph: the files with the most churn, the projects with the highest pipeline failure rates, and the code only one contributor has touched (bus factor risk). Cite real numbers and real project paths for every finding. Format it as one slide: three findings, one line each, metric plus where it lives. End with the graph totals you drew from: how many projects, files, and pipelines."

Look for high-churn files with a single contributor. The graph totals at the end are your proof the snapshot covers the whole group, not a sample.

6. Duplicate detection

"Using GitLab Orbit, my team is about to write a new [helper, client, or service]. Before we do, query the graph for existing implementations anywhere in our group: similar class and function names, similar dependencies, similar callers. List every candidate with its project, file path, and main consumers so I can judge reuse versus rebuild. Tell me how many projects the graph searched, so I know nothing was skipped."

Pick something an engineer built recently. **Look for** the existing implementation they didn't know about. Every match is duplicated work you can avoid.

Recording what you find

- The skill's report captures the prompt, both answers, and the measurements. Fill in the human verdict line and post it to the feedback epic (see the feedback guide).
- Testing manually? Keep the exact prompt, the answer, and your complete/grounded/fast scores for both runs.
- Bring the most surprising result (good or bad) to your weekly sync.
- Wrong or incomplete answers are high-value feedback. Flag them with the query you ran.

Orbit Design Partner Program · Orbit is in Beta · Agent setup (glab CLI first, MCP secondary): see the [Onboarding Guide](#)